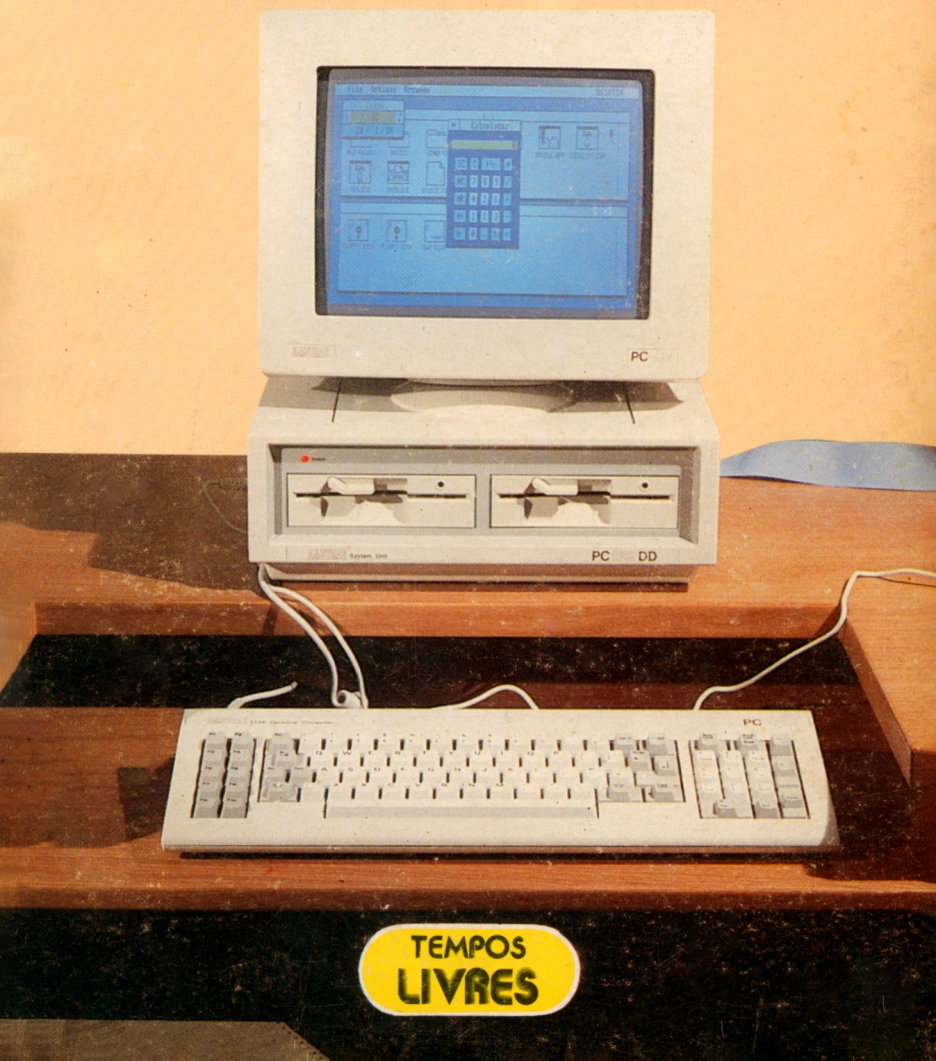
CLIVE GIFFORD e TIM HARTNELL

GUIA DE PROGRAMAÇÃO DO AMSTRAD



TEMPOS LIVRES

EDITORIAL
PRESENÇA

Rua Augusto Gil, 35-A
1000 LISBOA
Telef. 73 41 91

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Petar Trifunovitch* e *Sava Vukovitch*
4. ABC do Bridge, *Pierre Jais* e *H. Lahana*
5. Guia Prático de Fotografia, *W. D. Emanuel*
6. ABC do Judo, *E. J. Harrison*
7. Como Fazer Cinema, *Paul Petzold*
8. Bridge Moderno, *Pierre Jais* e *H. Lahana*
9. Fotografia — Técnicas e Truques I, *E. Smith*
10. ABC dos Estilos, da Arquitectura ao Mobiliário, *A. Aussel*
11. Fotografia — Técnicas e Truques II, *E. Smith*
12. A Pesca Submarina, *António Ribera*
13. Teoria dos Finais de Partida, *Yuri Averbach*
14. Aprenda Rádio, *B. Fighiera*
15. Guia do Cão, *L. L.-Robert* e *J.-P. Robert*
16. ABC do Aquário, *Anthony Evans*
17. Iniciação à Electricidade e Electrónica, *Fernand Huré*
18. Os Transístores, *Fernand Huré*
19. Karaté I, *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, *C. Péricone*
21. Construa o seu Receptor, *B. Fighiera*
22. Montagens Electrónicas, *B. Fighiera*
23. O Berbequim Eléctrico, *Villy Dreier*
24. Cactos, *J. Nilaus Jensen*
25. Iniciação à Alta Fidelidade, *Peter Turner*
26. O Aquário de Água Doce, *Paulo de Oliveira*
27. ABC do Ténis, *Fonseca Vaz*
28. Karaté II, *Albrecht Pflüger*
29. ABC da Criação de Canários, *C. Af Enehjelm*
30. Ginástica Feminina, *Sonja Helmer Jensen*
31. Cartomancia, *Rhea Koch*
32. Calculadoras Electrónicas de Bolso, *E. Dam Ravn*
33. O Pastor Alemão, *Gilles Legrand*
34. Xadrez — Teoria do Meio Jogo, *I. Bondarevsky*
35. Manual do Super 8 — I, *Myron A. Matzkin*
36. ABC da Criação de Periquitos, *C. H. Rogers*
37. O Livro dos Gatos, *B. Gerber* e *H. Bielfeld*
38. Manual do Super 8 — II, *Myron A. Matzkin*
39. ABC do Mergulho Desportivo, *Walter Mattes*
40. Circuitos Integrados/Aplicações Práticas, *F. Bergtold*
41. A Apicultura, *H. R. C. Riches*
42. ABC do Cultivo das Plantas, *H. G. W. Fogg*
43. ABC da Criação de Pombos, *Kai R. Dahl*
44. Construção de Caixas Acústicas de Alta Fidelidade, *R. Brault*
45. Raças de Canários, *Klaus Speicher*
46. Jogos de Cartas, *Graciano Dolma*
47. Cocker Spaniels, *H. S. Lloyd*
48. ABC da Caça, *Fabián Abril*
49. Aprenda Televisão, *Gordon J. King*
50. Iniciação à Pesca, *Juan Nadal*
51. Basquetebol, *Marius Norregard*
52. Cães de Caça, *Santiago Pons*
53. Aprenda Electrónica, *T. L. Squires* e *C. M. Deason*
54. A Avicultura, *Jim Worthington*
55. A Produção de Coelhos, *Ph. Surdeau* e *R. Hénaff*
56. ABC dos Computadores, *T. F. Fry*
57. Natação para Crianças, *John Idorn*
58. O Boxer, *Anni Mortensen*
59. Voleibol, *Ole Hansen e Per-Göran Persson*
60. Iniciação à Vela, *Donald Law*
61. ABC da Filatelia, *Jacqueline Caurat*
62. A Pesca à Beira-Mar, *J.-M. Böelle* e *B. Doyen*
63. Enxerto de Árvores de Fruto, *Alejo Rigau*
64. A Cultura do Morangueiro, *Luís Alsina Grau*
65. Emissores-Receptores (Walkies-Talkies), *P. Duranton*
66. Iniciação à Fotoelectrónica, *Heinz Richter*
67. Doces e Conservas de Fruta, *Robin Howe*
68. A Criação de Hamsters, *C. F. Snow*
69. A Criação de Porcos, *Roy Genders*
70. Calendário do Horticultor, *Luís Alsina Grau*
71. Jogos Electrónicos, *F. G. Rayer*
72. Cultivo de Cogumelos e Trufas, *Alejo Rigau*
73. Aprenda Televisão a Cores, *Gordon J. King*
74. Gravação em Fita Magnética, *Ian R. Sinclair*
75. Poda de Árvores e Arbustos, *Roy Genders*
76. Como Treinar o Seu Cão, *E. Fitch Daglish*
77. Instrumentos de Medida e Verificação, *H. Stöckle*
78. A Criação de Caracóis, *Matías Rosa*
79. Rádio — Fundamentos e Técnicas, *G. J. King*
80. Como Fazer Gelados, *Sylvie Thiébault*
81. Iniciação à Jardinagem, *Noel Clarasó*
82. A Congelação dos Alimentos, *S. Lapointe*
83. Windsurf — Prancha à Vela, *E. Prade*
84. Raças de Cães, *O. Hasselfeldt*
85. Rummy e Canasta, *Claus D. Grupp*
86. A Encadernação, *Annie Persuy*
87. Aprenda Electricidade, *Heinz Richter*
88. Taxidermia, Embalsamamento de Aves e Mamíferos, *Harry Hjortaa*
89. Jogging — Correr para Manter a Forma, *Werner Sonntag*
90. ABC da Cozinha Chinesa, *Sonya Richmond*
91. Jogos T.V., *C. Tavernier*
92. Amplificadores de Som, *Richard Zierl*
93. O Livro do Poker, *Claus D. Grupp*
94. Aprenda a Desenhar, *Rose-Marie de Prémont* e *Nicole Philippi*
95. O Minitrampolim na Escola, *Sonja Helmer Jensen e Klaus Dano*
96. Jogos de Luzes e Efeitos Sonoros para Guitarras, *B. Fighiera*
97. O Cultivo do Tomate, *Louis N. Flawn*
98. Pilhas Solares, *F. Juster*
99. A Criação Doméstica de Coelhos, *C. F. Snow*
100. Iniciação ao Futebol, *W. Männle* e *H. Arnold*
101. Horóscopos Chineses, *Georg Haddenbach*
102. Guia Prático de Marcenaria, *C. H. Hayward*
103. Andebol, *Fritz* e *Peter Hattig*
104. Dispositivos Anti-Roubo, *H. Schreiber*
105. Perus, Pintadas e Codornizes, *Jérome Sauze*
106. Crepes — Doces e Salgados, *Florence Arzel*
107. Aperitivos e Entradas, *Myrette Tiano*
108. Ténis de Mesa, *Leslie Woollard*
109. Aprenda Surf, *R. Abbott* e *M. Baker*
110. Futebol — Técnica e Táctica, *Kurt Lavall*
111. A Vaca Leiteira, *Colin T. Whittemore*
112. O Cubo Mágico, *Josef Trajber*
113. O Perdigueiro Português, *José M. Correia*
114. Pizzas e Massas à Italiana, *Marieanne Rank*
115. O Cubo para Quem Já o Faz, *Josef Trajber*
116. A Pirâmide Mágica, A Torre, O Barril do Diabo, *M. Mrowka-W. J. Weber*
117. Gansos e Patos, *Marie Mourthe*
118. Iniciação ao Kung-Fu, *A. P. Harrington*
119. Electrónica e Fotografia, *Hanns-Peter Siebert*
120. O Livro da Fortuna, *Douglas Hill*
121. Construção de um Alimentador de Corrente, *W. Baitinger*
122. Hóquei em Patins, *Francisco Velasco*
123. Técnicas de Tiro, *Anton Kovacic*
124. Aprenda a Tricotar, *Uta Mix*
125. ABC da Patinagem, *C.-M.* e *R. Kerler*
126. A Pesca e os Seus Segredos, *A. Deschamps*
127. O Osciloscópio, *R. Rateau*
128. Guia Prático da Banda do Cidadão, *T. M. Normand*

129. Sumos e Batidos, *Manfred Donderski*
130. Introdução à Programação de Microcomputadores, *P.C. Sanderson*
131. Aprenda Croché, *Uta Mix*
132. ABC do Microprocessador, *P. Mélusson*
133. Guia Prático de Basic, *Roger Hunt*
134. Introdução à Electrónica Digital, *Ian R. Sinclair*
135. ABC do Vídeo, *David K. Matthewson*
136. Fotografia em Movimento, *Don Morley*
137. Guia de Cobol, *Ray Welland*
138. Fotografia a Pequena Distância, *S. F. Ray*
139. Guia Moderno da Canaricultura, *Manuel Gonçalves*
140. Minielectrónica para Amadores, *H. Richter*
141. ABC da Programação de Computadores, *John Shelley*
142. Tarot — O Futuro Pelas Cartas, *E. J. Nigg*
143. ABC da Equitação, *Dorothy Johnson*
144. Como Programar o Seu ZX 81, *P. Gueulle*
145. 100 Avarias TV e a Maneira Prática de as Detectar, *P. Duranton*
146. ABC da Horticultura, *Louis Giordano*
147. Basic para Microcomputadores, *A. P. Stephenson*
148. Como Programar o Seu ZX Spectrum, *Tim Hartnell* e *Dilwyn Jones*
149. Iniciação aos Motores Diesel, *D. S. Maclean*
150. 60 Jogos para o ZX Spectrum, *D. Harwood*
151. As Linhas da Mão, *Rose Hubert*
152. Cozinha Italiana, *Rotraud Degner*
153. Manual do ZX Spectrum, *R. J. Simpson* e *T. J. Terrell*
154. Z80 Assembler para o ZX Spectrum — Iniciação ao Código de Máquina, *J. P. Fragoso*
155. Aeróbica, *H. Schulz*
156. ABC do Atletismo, *Denis Watts*
157. 26 Programas Basic para Microcomputadores, *Derrick Daines*
158. Aprenda Pascal no Seu Microcomputador, *Jeremy Ruston*
159. Guia Moderno da Suinicultura, *Colin Whittemore*
160. O Bar em Sua Casa — 888 Cocktails, *Aladar von Wesendonk*
161. Código de Máquina para Principiantes, *J. Walsh*
162. Código de Máquina para Programadores Avançados, *P. Holmes*
163. ABC da Fruticultura, *Henri Gosselin*
164. ABC da Canoagem, *Alan Byde*
165. Guia de Fortran, *Philips Ridler*
166. Manual da Secretária, *Philippa Ramage*
167. ABC das Antenas, *Gordon J. King*
168. Programar Aventuras no Seu Computador, *A. Nelson*
169. Guia do Sinclair QL, *Boris Allan*
170. Novas Aventuras no Seu ZX Spectrum, *P. Shaw* e *J. Mortleman*
171. O Computador no Escritório, *John Shelley*
172. Sobremesas, *Fred Timber*
173. Rádio — Do Circuito Oscilante ao Receptor de Ondas Curtas, *Richard Zierl*
174. Xadrez: ABC das Aberturas, *V.N. Panov*
175. A Construção de Pequenos Transformadores, *M. Dorian* e *F. Juster*
176. Guia de Pascal, *David Watt*
177. O ZX Spectrum na Educação, *Tim Hartnell, Christine Johnson* e *David Valentine*
178. Iniciação ao Snooker, *John Pulman*
179. Circuitos com Diacs, Tiristores e Triacs, *Fritz Bergtold*
180. O Jogo do Mah-Jong, *Ursula Eschenbach*
181. Cultura Hidropónica, Culturas sem Solo, *Laura Fronty*
182. 49 Jogos Explosivos para o ZX Spectrum, *T. Hartnell, D. Perry, G. Charlton, N. Pellinacci* e *M. Young*
183. Iniciação ao Golfe, *J. C. Jessop*
184. Como se Constrói um Receptor de FM, *Richard Zierl*
185. Iniciação à Navegação Marítima, *A. Vieira*
186. Processamento de Dados, *T. F. Fry*
187. O ZX Spectrum na Gestão de Pequenas Empresas, *Luís de Campos*
188. Programar Grafismos no Seu Computador, *Michel Rousselet*
189. O IBM PC e Sistemas Compatíveis, *Carlos Reis e João Capaz*
190. Antenas para a Banda do Cidadão, *P. Gueulle*
191. Paciências, *Irmgard Wolter*
192. Defesa Pessoal, *Syd Hoare*
193. Processamento de Texto, *Lew Hollerbach*
194. Inteligência Artificial no Seu Spectrum e Spectrum +, *Tim Hartnell*
195. Manual do Correspondente de Inglês, *W. G. Tackle* e *Gomes Pitta*
196. Como Interpretar um Balanço, *Carlos Nabais*
197. Iniciação à Tecnologia da Informação, *Garry Marshall*
198. Manual de Programação Avançada do ZX Spectrum, *John Lettice*
199. O Espargo, *Mário F. B. Ripado*
200. Computador Atari, Manual Completo do Utilizador, *Luís de Campos*
201. Manual de Guitarra Eléctrica, *Giorgio Onetti* e *Giovanni Gaglio*
202. A Produção de Frangos, *Ph. Surdeau* e *R. Hénaff*
203. Como Construir um Telescópio, *J. Garcia*
204. Aventuras Gráficas para o Spectrum 48K, *R. G. Hurley*
205. Receitas de Produtos de Beleza Naturais, *Liz Sanderson*
206. Filmar com Câmara Vídeo, *Frederic W. Rosen*
207. Guia de Programação do Amstrad, *Clive Gifford* e *Tim Hartnell*

CLIVE GIFFORD
TIM HARTNELL

GUIA DE PROGRAMAÇÃO DO AMSTRAD

Para Amstrad CPC 464, CPC 664 e CPC6128
e PCW8256, PCW8512 e PC1512

EDITORIAL PRESENÇA

FICHA TÉCNICA

Título original: *The Amstrad Programmer's Guide*
Autores: *Clive Gifford e Tim Hartnell*
© *Clive Gifford and Tim Hartnell 1985*
Edição publicada por acordo com Pitman Publishing Limited, London
Tradução: *Eduardo Nogueira*
Tradução © by Editorial Presença, Lisboa, 1987
Capa: *Fotografia de Manuel Palma*
Fotocomposição: *TEXTO E LINHA - Estúdio Gráfico, Lda.*
Impressão e acabamento: *Rolo & Filhos, Mafra*
Tiragem: 3.000 exemplares
1.ª edição, Lisboa, 1987

Reservados todos os direitos
para a língua portuguesa à
EDITORIAL PRESENÇA, LDA.
Rua Augusto Gil, 35-A 1000 LISBOA

NOTA DA EDIÇÃO PORTUGUESA

Agradece-se ao Sr. Eurico da Fonseca a gentileza em ter colaborado graciosamente na redacção dos capítulos sobre os PCW8256/8512 e PC1512, que constituem um indispensável aditamento a esta edição, dado que acompanham os mais recentes desenvolvimentos da gama de computadores Amstrad.

AGRADECIMENTOS

Os autores gostariam de agradecer a todos aqueles que ajudaram a produzir este livro, especialmente a Catherine, Betty, Bill e Al.

PREFÁCIO

Bem-vindo ao Amstrad! O leitor deve estar bastante satisfeito com o seu novo computador — é uma máquina bastante avançada e oferece vantagens não existentes noutros computadores domésticos mais caros.

Este livro foi escrito para tratar de tudo aquilo que falta no manual do Amstrad, e de muito mais. E, depois de o leitor dominar a linguagem BASIC, poderá aproveitar o que dizemos nos últimos capítulos sobre a forma de melhorar os seus programas, dando-lhes uma aparência mais profissional, mais acção e estrutura.

Antes de ter decidido que preferia um micro Amstrad, o leitor deve ter pensado no que desejaria fazer com ele. Talvez pretendesse apenas jogar um pouco (e o Amstrad é certamente bom em jogos) ou o quisesse utilizar em tarefas mais sérias (escrever cartas, resolver problemas complexos, gerir as suas finanças, educar crianças). Talvez quisesse aprender a programar e penetrar no segredo que rodeia os computadores — aprender o que podem ou não fazer. Todas estas razões justificam plenamente a compra de um computador, e verificará que sobre todas elas diremos alguma coisa ao longo deste volume.

Leia este livro, divirta-se com os programas incluídos e aprenda a programar o Amstrad como um profissional!

Clive Gifford, Tim Hartnell
Maio 1985

1

COMO FUNCIONA O SEU COMPUTADOR?

Na nossa vida quotidiana trabalhamos num sistema numérico baseado em 10 — sistema decimal. Possuímos dez algarimos (incluindo o zero), e com eles podemos representar qualquer número. A posição do algarismo indica o seu valor, sendo tal posição dependente do número de zeros. Sabemos assim que 90 é dez vezes maior do que 9, porque o zero esclarece o valor do algarismo «significativo», o 9.

Os computadores trabalham normalmente num sistema numérico baseado no 2 — o sistema binário. Este utiliza apenas uns e zeros, sendo o valor de cada algarismo indicado pela sua posição na sequência. Vejamos a correspondência entre os números 1 a 9 no sistema decimal e os equivalentes valores binários:

1	2	3	4	5	6	7	8	9
1	10	11	100	101	110	111	1000	1001

Como se pode verificar, os números binários crescem bastante depressa, sendo difícil compreender imediatamente o seu valor. Por que razão se usam números tão estranhos? A resposta é simples, um computador é formado essencialmente por uma vasta colecção de interruptores, cada um dos quais, como é habitual, só pode ter dois valores — «ligado» ou «desligado». Se usarmos o zero para representar «desligado» e o um para representar «ligado», podemos representar qualquer número desde que utilizemos uma quantidade apropriada de interruptores.

O Amstrad, como é evidente, não nos obriga a conhecer a aritmética binária que ele próprio usa. Realiza as conversões necessárias entre o nosso sistema numérico e o dele. A maior parte dos computadores utiliza a aritmética binária. Nos primeiros tempos da programação, os profissionais eram obrigados a trabalhar com zeros e uns, o que significava que a escrita de um programa consumia muito tempo e consti-

tuía uma tarefa frustrante. Qualquer erro de programa obrigava a verificar um elevado número de zeros e uns.

Nesses tempos, os programadores eram de facto uma raça à parte. A tensão intelectual a que se encontravam expostos devia ser imensa. Felizmente, o computador faz hoje a maior parte do trabalho, transformando cadeias de zeros e uns em números que podemos compreender.

Mas o que acabamos de dizer, se bem que nos indique como um computador manipula os números, não nos ajuda a compreender como «pensa».

A lógica do computador, a sua capacidade para tomar decisões, baseia-se nas descobertas de um matemático inglês de princípios do século, George Boole. O resultado da sua obra é conhecido pelo nome de «álgebra booleana».

O valor das descobertas de Boole reside no facto de ter reduzido a tomada de decisões a um processo matemático — um processo que demonstrou ser muito fácil de usar em computadores. O seu Amstrad guarda «verdadeiro» e «falso» sob a forma de números.

Ilustremos esta informação. Se ligar o seu computador e escrever PRINT 1 = 1 (uma afirmação verdadeira), verá o computador imprimir −1, mostrando que considera «verdadeiro» igual a −1. Escreva agora uma afirmação falsa, como PRINT 3 = 4. Desta vez o computador escreverá zero, indicando que considera uma afirmação falsa como igual a zero.

Lógico, meu caro Watson

Como veremos adiante, o Amstrad pode tomar decisões que envolvem «operadores lógicos». Ou seja, pode verificar se a afirmação A e a afirmação B são verdadeiras, se apenas uma delas o é, ou se são ambas falsas. O computador pode até tratar cadeias longas e complexas de afirmações lógicas, por exemplo determinando se a instrução A e a B são verdadeiras, e a C e a D ou a E falsas. O suporte deste poder de tomada de decisões do computador, diferente da capacidade de manipular números, reside na sua capacidade de tratamento da álgebra booleana e de traduzir as suas descobertas — expressas matematicamente — numa resposta que nós somos capazes de compreender.

Isto não esgota o assunto. Observemos melhor o nosso Amstrad. Dispõe de um modo de aceitar informação proveniente do mundo exterior (através do teclado e de outros dispositivos como o *joystick*) e de um meio de tradução desta informação nos uns e zeros de que necessita.

Possui um método de armazenar essa informação até necessitar dela, de juntar a informação nova aos factos que já conhece e de dar saída (através do visor ou da impressora) à informação que contém.

Se, por exemplo, quisermos que o nosso Amstrad some quatro números, poderemos dizer-lhe primeiro que a letra A representa o número 4, que a letra B é igual a 5, que C é igual a 9 e D igual a 2. Aceitará esta informação, guardando-a sob a forma de um grupo de algarismos binários. Depois podemos dizer-lhe que some A e B, e depois C e D, e imprima os resultados.

Quando se adquire o computador, este contém já algumas instruções, que lhe «dizem» como deve somar dois números, por exemplo, e como nos pode indicar o resultado.

Em seguida podemos querer determinar qual dos dois resultados (A + B ou C + D) é maior, e imprimir esta informação. A capacidade para fazer isto encontra-se também incorporada no computador sob a forma de um conjunto de zeros e uns. Depois de o computador descobrir qual dos dois resultados é maior, podemos indicar-lhe outra tarefa.

O leitor poderá ler de novo esta explicação bastante simples até a compreender melhor. Poderá então começar a aperceber-se da forma como o seu Amstrad trabalha. De facto, não necessita de saber como um computador trabalha para o poder utilizar — tal como não é preciso saber como funciona a televisão para podermos ver um filme —, mas se tivermos uma ideia sobre o assunto poderemos utilizar melhor o «génio» electrónico ao nosso dispor.

2

PRIMEIROS PASSOS NA PROGRAMAÇÃO

A linguagem BASIC usada no seu Amstrad é bastante fácil de dominar. O leitor conseguirá certamente programar um pouco daqui a meia hora, mesmo que nunca tenha programado antes. Mas certamente só dominará completamente a tarefa de programação daqui a algum tempo.

Estudaremos os comandos disponíveis no seu computador seguindo uma rotina muito simples. Em primeiro lugar, daremos uma breve descrição de cada palavra que o computador reconhece, mostrando em seguida a forma de a utilizar. Apresentaremos alguns programas curtos, combinando as palavras novas com as já conhecidas; deste modo, poderemos ver como é possível usar palavras diferentes para construir um programa.

De vez em quando incluiremos programas maiores; aliás, o leitor pode até introduzi-los na máquina desde já. Mesmo que não compreenda exactamente como funcionam estes programas, pode mesmo assim ter o prazer de os usar.

Linhas

Todos os programas em BASIC (a linguagem de programação que vai começar a aprender e que está instalada em praticamente todos os computadores pessoais que se fabricam no mundo) são compostos por uma série de linhas, cada uma das quais se inicia por um número. Em termos gerais, o computador executa um programa por ordem numérica das linhas que o constituem, começando pelo número mais baixo e prosseguindo em direcção ao mais alto. De vez em quando, o computador saltará para uma linha fora da ordem (usando instruções como a GOTO), mas isto será tratado noutro ponto do livro.

Daqui em diante, consideraremos que o leitor ligou já o seu Amstrad, e que vai introduzir nele cada um dos programas a que nos referi-

remos. Verificará que ganha muito mais com este livro e que aprende a programar muito mais rapidamente se introduzir estes programas, experimentando todos os exercícios descritos. Este texto permite-lhe aprender por si mesmo, mas deverá aplicar-se um pouco para obter resultados.

PRINT, LIST, NEW, RUN

PRINT é uma das palavras da linguagem BASIC mais usadas, e significa exactamente aquilo que quer dizer («imprimir»); uma instrução PRINT indica ao computador que deve imprimir alguma coisa no visor. A palavra PRINT e os outros comandos que estudaremos nesta secção do livro são palavras que o computador já conhece no momento em que você o compra.

Para ilustrarmos os resultados da instrução PRINT, escreva PRINT «TESTE» (não se esqueça de escrever as aspas antes e depois do texto que pretende imprimir), e carregue na tecla ENTER. Verá então a palavra TESTE surgir por baixo da linha PRINT «TESTE». Mais abaixo verá o termo Ready, indicando que o computador fez aquilo que lhe pediu e espera agora novas instruções.

Escreva o seguinte conjunto de instruções (um programa), carregando em ENTER sempre que complete uma linha:

```
10 PRINT ''A''
20 PRINT ''BELA''
30 PRINT ''AMÉRICA''
```

Depois de ter introduzido este programa no computador, escreva a palavra RUN e carregue de novo em ENTER. Como já verificou, carrega-se nesta última tecla sempre que se pretende que o computador aceite a ordem dada. A instrução RUN põe o programa em funcionamento, e o leitor deverá observar no visor o seguinte texto:

```
A
BELA
AMÉRICA
```

Podemos já aprender bastantes coisas a propósito deste pequeno programa. Como se referiu anteriormente, as linhas de programa são numeradas e o programa tende a ser executado a partir da primeira linha e avançando em direcção à última. Não há necessidade de numerar as linhas do programa de dez em dez, começando na linha 10, mas será bom que adopte este hábito; se o fizer, disporá de espaço para escrever novas linhas entre as já existentes, em caso de necessidade.

A instrução PRINT determina o modo como a informação será impressa no visor. Altere a primeira linha do programa de modo a ter a aparência seguinte:

```
10 PRINT, "A"
```

Alteram-se linhas escrevendo de novo toda a linha com as alterações requeridas e carregando em ENTER; a nova linha substituirá então a linha anterior (existem outras formas de modificar as linhas do programa, que não obrigam a reescrever toda a linha, mas deixá-las-emos para mais tarde).

Depois de escrever a nova linha, e de executar o programa usando RUN, notará que o resultado passou a ser o seguinte:

```
                    A
BELA
AMÉRICA
```

Conclui-se portanto que a vírgula serve para deslocar a posição de impressão mais para a direita.

Podemos usar portanto a vírgula para formatar a impressão. Por exemplo, podemos alinhar uma fiada de números num dos lados do visor.

Eliminemos agora o programa anteriormente escrito escrevendo a palavra NEW. Depois de carregarmos, como é habitual, na tecla ENTER, todo o programa anterior é apagado da memória da máquina.

Podemos verificar se o computador está vazio escrevendo a palavra LIST e carregando uma vez mais em ENTER. Veremos que a memória do Amstrad está vazia porque não «lista» nenhum programa. Discutiremos a instrução LIST daqui a pouco.

Antes disso, no entanto, escreva e execute (usando RUN) o programa seguinte:

```
10 PRINT "ISTO É"
20 PRINT "UM TESTE"
```

Verá escrito no visor o texto seguinte:

```
ISTO É
UM TESTE
```

Altere agora a linha 10 para o formato indicado em seguida:

```
10 PRINT "ISTO É ";
```

não esquecendo um espaço entre a letra É e as aspas. Execute de novo o programa, e deverá obter um resultado igual ao seguinte:

ISTO É UM TESTE

Como se pode verificar, as palavras encontram-se agora numa mesma linha. Isto deve-se ao facto de o ponto e vírgula juntar o final de uma instrução PRINT à que se segue.

TAB e LIST

A instrução TAB (de «tabular») permite-nos definir de modo rigoroso a posição de impressão. TAB é escrito a seguir à palavra PRINT, e acompanhada de um número escrito entre parêntesis. É este número que define a quantidade de espaços que serão deixados em branco entre o início da linha e o princípio do texto nela impresso. Vejamos alguns exemplos:

```
10 PRINT TAB(2);"2"
20 PRINT TAB(7);"7"
30 PRINT TAB(14);"14"
40 PRINT TAB(32);"32"
```

A variável de controlo, o número escrito dentro dos parêntesis, pode ser de facto uma variável em vez de um simples número, como se vê no programa seguinte:

```
10 FOR J = 1 TO 24
20 PRINT TAB(J);J
30 NEXT J
```

O termo «variável» e o modo de usar a instrução FOR aqui apresentada na linha 10 serão discutidos mais adiante. O resultado da execução do programa anterior é o apresentado na figura 1.

Depois de introduzir este programa na memória do seu Amstrad, pode consultá-lo escrevendo LIST e carregando na tecla ENTER. Experimente... e verá a listagem do programa aparecer no visor.

LIST pode também ser usada para obter apenas parte da listagem. Se dispomos de um programa com dez linhas, por exemplo, numeradas de dez em dez (sendo, portanto, 10, 20, 30, etc., até 100), o Amstrad permite listar apenas as últimas quatro linhas escrevendo o seguinte:

LIST 70 —

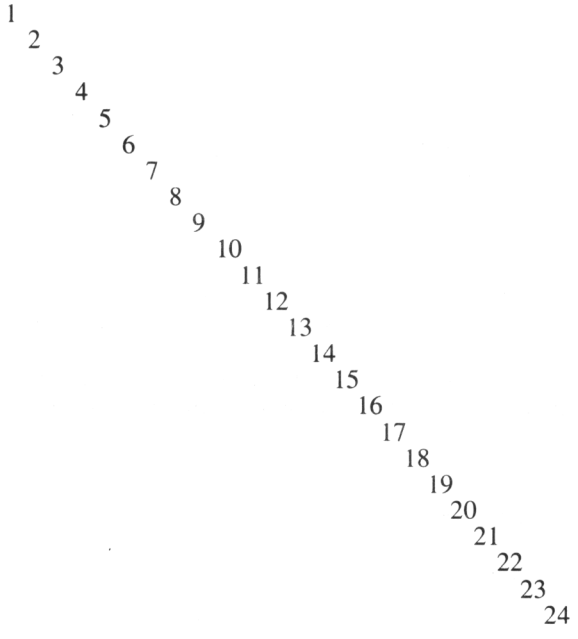


Figura 1

Isto lista todas as linhas que se seguem ao hífen; neste exemplo, o computador listará as linhas 70, 80, 90 e 100. Do mesmo modo, para obter todas as linhas até um dado número, pode-se escrever:

LIST — 40

Esta instrução listará todas as linhas desde o início do programa até à especificada (inclusive) — o que significa que no caso considerado seriam listadas as linhas 10, 20, 30 e 40.

Se as linhas que desejamos estudar não se encontram no início ou no final do programa, podemos observá-las escrevendo a instrução com o seguinte formato:

LIST 20 — 60

Esta instrução listará as linhas 20, 30, 40, 50 e 60.

Variáveis

Quase todos os programas que o leitor virá a escrever ou ver escritos utilizarão variáveis. As variáveis são letras ou combinações de letras e números, começando sempre por uma letra. Recebem valores durante a execução do programa, equivalendo de facto a esses números nos cálculos efectuados.

Para esclarecermos um pouco melhor o que acabámos de dizer, observemos o segmento de programa seguinte:

```
10 A = 36
20 B = 9
30 C = A+B
40 PRINT A;" + ";B;" = ";C
```

Neste programa é atribuído um valor de 36 à variável A (que passa a actuar como equivalendo a esse valor ao longo de todo o programa), um valor de 9 à variável B, e um total de A+B à variável C (como se pode verificar na linha 40). As variáveis usadas deste modo são chamadas variáveis numéricas. Podem alterar os respectivos valores ao longo do programa, como se mostra no seguinte exemplo:

```
10 A = INT (RND*10) + 1
20 B = INT (RND*10) + 1
30 PRINT "A é ";A
40 PRINT "B é ";B
50 FOR T = 1 TO 500:NEXT T
60 PRINT
70 GOTO 10
```

Se bem que uma variável possa ser representada por qualquer letra entre A e Z, os nomes das variáveis não são restringidos a letras isoladas. Qualquer combinação de letras e números, desde que comecem por uma letra, é um nome aceitável, pelo que C2PO, R2D ou PESCADOR são nomes válidos.

O leitor verificará que o uso de nomes de variáveis explícitos pode ajudar a tornar os programas mais fáceis de compreender. Por exemplo, no programa que se segue, que determina uma percentagem, o número maior é atribuído a uma variável chamada NUMEROGRANDE, e o mais pequeno à variável NUMEROPEQUENO; o resultado do cálculo é passado a uma variável chamada PERCENTAGEM.

```
10 NUMEROGRANDE = 761
20 NUMEROPEQUENO = 234
```

```
30 PERCENTAGEM = NUMEROPEQUENO*100/NUMERO-
GRANDE
40 PRINT "NUMEROGRANDE = ";NUMEROGRANDE
50 PRINT "NUMEROPEQUENO = ";NUMEROPEQUENO
60 PRINT "PERCENTAGEM = ";PERCENTAGEM;"%"
```

Como se pode ver, é muito fácil compreender o que se passa. É improvável que o leitor use nomes tão compridos nos seus programas, mas nada nos impede de utilizar versões abreviadas de nomes como os acima indicados, desde que sejam ainda suficientemente claros.

Cadeias

Até agora temos falado apenas de variáveis numéricas... mas existem também *variáveis de cadeia* (*string variables*). Em jargão de computador, uma cadeia é qualquer coisa encerrada dentro de aspas. Pode ser usada por exemplo numa instrução PRINT, como já vimos anteriormente.

Vejamos portanto exemplos de cadeias:

```
"O SOL ESTÁ QUENTE"
"A RESPOSTA PARA O PROBLEMA É"
```

A única diferença entre a variável que representa uma cadeia e a que representa um número é que o nome da variável de cadeia termina por um cifrão. Exemplos de nomes válidos de cadeias:

```
A$    FACE$    D123456$    G6H4$
```

Vejamos um programa que demonstra o uso de variáveis de cadeia:

```
10 N$ = "NARIZ"
20 D$ = "O MEU CÃO"
30 B$ = "NÃO"
40 C$ = "TEM"
50 PRINT
60 PRINT D$
70 PRINT B$;C$
80 PRINT N$
```

Note-se que apesar de a cadeia dever encontrar-se geralmente encerrada em aspas, o nome da variável não utiliza aspas, mesmo quando usado numa instrução PRINT.

O leitor notará que as variáveis numéricas são muito usadas nos programas da primeira parte deste livro. As variáveis de cadeia não são tão comuns. As cadeias e a manipulação de cadeias são discutidas em pormenor um pouco mais adiante, mas o leitor já sabe pelo menos reconhecê-las.

3

MAIS PROGRAMAÇÃO

Vamos agora acrescentar alguns outros comandos BASIC ao repertório do leitor. No final deste capítulo já conhecerá as instruções CLS, REM, CONT, GOTO, ON... GOTO, GOSUB, ON... GOSUB, PAPER, PEN e LOCATE.

Começando pela instrução mais simples, CLS, é usada para limpar o visor. Acrescente uma linha com CLS no início de um dos programas do capítulo anterior e verá os resultados.

O leitor estará já a interrogar-se sobre o paradeiro das 27 cores do Amstrad. O Capítulo 15, dedicado aos grafismos, contém os pormenores apropriados, mas entretanto podemos já apresentar algumas noções sobre o assunto.

O Amstrad utiliza três modos gráficos e, quando ligado, passa imediatamente ao modo 1 (25 linhas de 40 caracteres com um máximo de 4 cores). Para alterar a cor de fundo, use a instrução PAPER seguida de um número entre zero e três. A cor de fundo automática, quando a máquina é ligada, é a zero (azul-escuro). Depois de alterar a cor de fundo, limpe o visor escrevendo CLS.

A instrução PEN funciona do mesmo modo que a PAPER, seleccionando uma das quatro cores disponíveis e usando-a como cor do primeiro plano. Não se usa CLS quando se altera a PEN.

Vejamos um exemplo do funcionamento de PEN e PAPER; depois de ter introduzido o programa, escreva RUN e observe o efeito de todas as combinações possíveis das quatro cores disponíveis no Amstrad (vermelho, azul-escuro, azul-claro e amarelo). Note-se que se pode alterar estas cores usando a instrução INK, explicada mais claramente no Capítulo 15.

```
10 ' PEN E PAPER'
20 '
30 FOR ATRÁS = 0 TO 3
40 FOR FRENTE = 0 TO 3
```

```
50 PEN FRENTE:PAPER ATRÁS
60 CLS
70 PRINT "PEN NÚMERO";FRENTE
80 PRINT
90 PRINT "SOBRE PAPEL NÚMERO";ATRÁS
100 FOR B = 1 TO 1500 : NEXT
110 NEXT : NEXT
120 PAPER 0 : PEN 1
```

Observando este programa, verificamos que as duas primeiras linhas se iniciam por um apóstrofo, uma forma abreviada da instrução REM.

A palavra REM indica «observação» (REMARK). Refere-se a uma palavra ou frase escrita no interior do programa a fim de identificar a função das instruções que se seguem. As instruções REM são ignoradas pelo computador, pelo que podemos usá-las para guardar qualquer informação que nos possa auxiliar a entendermos o programa.

O leitor verificará que as instruções REM são particularmente úteis quando voltamos a mexer num programa depois de uma longa interrupção. Talvez pense que nunca irá esquecer o que fazem as várias partes dos seus programas, e esta crença permanecerá até à primeira vez que tentar descobrir o que fez num programa no qual não toca há alguns dias. Daí em diante começará certamente a usar instruções REM...

Vejamos alguns exemplos de instruções REM:

```
80 REM FIM DO JOGO, IMPRESSÃO DA PONTUAÇÃO
100 REM+++PÁGINA DE TÍTULO+++
```

Se bem que possa parecer que as instruções REM apenas servem para perder tempo em programas pequenos, o leitor verificará que são imprescindíveis em programas compridos. Vale certamente a pena habituar-se a usar instruções REM.

Corrida de cavalos

Estudaremos agora várias instruções, usando um programa para mostrar os resultados de cada uma delas. O programa é chamado «Corrida de Cavalos», porque nele os números 1 a 4 correm de uma ponta à outra do visor, como se fossem quatro pequenos cavalos.

No início do programa, pensado para duas pessoas (se bem que seja fácil criar uma versão em que uma só pessoa jogue contra o computador), cada jogador possui 30 dólares. Cada jogador aposta 5 no cavalo que pensa ganhar, indicado pelo respectivo número. Se o cavalo ganha, o jogador recebe 10 dólares.

O jogo continua, em sucessivas corridas, até um dos jogadores deixar de ter dinheiro.

```
10 REM CORRIDA DE CAVALOS
20 X(1)=30: X(2)=30
30 BORDER 9:INK 0,9 : INK 1,0
40 INK 2,26
50 PAPER 0
60 CLS
70 FOR T = 1 TO 2
80 X(T)=X(T)—5 : IF X(T) <1 THEN 620
90 PRINT : PRINT
100 PEN 2
110 PRINT ''JOGADOR''; T; ''::''
120 PRINT ''TEM $'' ;X(T)
130 PRINT : PRINT
140 PRINT ''SOPRO DE VENTO                    = 1''
150 PRINT ''SORTE                             = 2''
160 PRINT ''VINHO TINTO                       = 3''
170 PRINT ''BELEZA NEGRA                      = 4''
180 PRINT : PRINT
190 PRINT ''INDIQUE O NÚMERO DO CAVALO EM QUE QUER
    APOSTAR (1—4)''
200 A$ = INKEY$
210 IF A$< ''1'' OR A$> ''4'' THEN 200
220 A = VAL (A$)
230 Z(T) = A
240 IF T = 2 AND Z(1) = Z(2) THEN PRINT ''O NÚMERO 1
    TEM'';Z(1): FOR D = 1 TO 1000 :  NEXT : GOTO 90
250 NEXT
260 CLS
270 PEN 3
280 PRINT ''CORRIDA DE CAVALOS''
290 PEN 2
300 PRINT : PRINT ''/////////////////////////////////////''
310 LOCATE 1,10
320 PRINT ''/////////////////////////////////////''
330 PEN 3
340 FOR T = 1 TO 6 : LOCATE 34,T + 3 : PRINT '';'': NEXT
350 LOCATE 1,5
360 PEN 1
370 FOR T = 1 TO 4 : A(T) = T/32 : NEXT
380 FOR B = 1 TO 4
390 A(B) = A(B) + INT (RND*2)
```

```
400 PRINT TAB (A(B));" ";B;
410 IF B <4 THEN PRINT
420 IF A(B) >32 THEN PRINT : GOTO 470
430 NEXT B
440 LOCATE 1,5
450 GOTO 380
460 SOUND 1,180,100 : SOUND 1,120,100
470 CLS
480 PEN 3
490 PRINT "O VENCEDOR FOI" ;B
500 PRINT : PRINT
510 IF Z(1) = B THEN PRINT "E O JOGADOR 1 GANHOU":
    X(1) = X(1) + 10
520 IF Z(2) = B THEN PRINT "E O JOGADOR 2 GANHOU":
    X(2) = X(2) + 10
530 PEN 1
540 FOR T = 1 TO 900 : NEXT
550 PRINT "PREPAREM-SE..."
560 FOR T = 1 TO 900 : NEXT
570 PRINT " . . . PARA UMA NOVA  . . . "
580 FOR T = 1 TO 900 : NEXT
590 PRINT " . . . . . . . CORRIDA!!"
600 FOR T = 1 TO 2000 : NEXT
610 GOTO 60
620 PAPER 1 : PEN 3 : BORDER 2
630 PRINT : PRINT "O JOGADOR ";T
640 PRINT "ESTÁ FALIDO, PELO QUE "
650 PRINT "O VENCEDOR É O JOGADOR ";
660 IF T = 1 THEN PRINT 2 ELSE PRINT 1
670 END
```

Como este é o primeiro programa de dimensões razoáveis que o leitor introduz no seu computador, notou certamente algumas instruções que ainda não conhece. Algumas serão explicadas nas páginas que se seguem, e outras mais tarde. Depois de se ter divertido um pouco com o programa, convirá observar as linhas que o constituem a fim de descobrir o que fazem essas instruções. Volte depois ao texto, verificando se as suas deduções estavam correctas.

Algumas instruções novas

CONT

A primeira instrução que observaremos é CONT, que indica «continuar». É usada quando por alguma razão desejamos parar o programa e voltar depois a retomar a sua execução. Para ilustrar este comando, executemos novamente o programa e carreguemos em seguida duas vezes na tecla ESC. Será impressa uma mensagem de erro (do tipo "BREAK IN 430") e o programa pára. Se escrevermos em seguida CONT, o programa continuará (apesar de a mensagem de erro se manter no visor) a partir do ponto onde foi interrompido.

LOCATE

A instrução LOCATE é usada várias vezes no programa «Corrida de Cavalos», particularmente na parte que desenha a corrida propriamente dita.

Esta instrução serve para posicionar o cursor de impressão. Se já utilizou outro computador antes do seu Amstrad, reconhecerá em LOCATE o equivalente a PRINT AT ou PRINT @. A palavra LOCATE é seguida de dois números, que correspondem às coordenadas horizontal e vertical da nova posição do cursor.

No modo 1, aquele em que nos encontramos neste momento, o primeiro número (o valor na horizontal) pode variar entre 1 a 40, e o segundo (o valor na vertical) pode variar entre 1 e 25. O programa que se segue mostra o funcionamento de LOCATE:

```
10 CLS
20 PEN 1
30 INK 1,2,24
40 LOCATE 10,10
50 PRINT "X"
60 PEN 2
70 LOCATE 8,23
80 PRINT "X MARCA A POSIÇÃO 10,10"
```

GOTO

Esta instrução («ir para») tem um significado e uma função óbvios. Já foi dito anteriormente que os programas em BASIC tendem a ser executados pela ordem das linhas, começando pela primeira. GOTO é uma das instruções que pode alterar esta ordem de execução. Quando o computador encontra a ordem GOTO, passa para o número de linha indicado a seguir a esta palavra.

Voltando à nossa «Corrida de Cavalos», parte deste programa teve de ser repetido várias vezes (a rotina que desenha os cavalos). A instrução GOTO da linha 450 indica à máquina que deve saltar para a linha 380, o início da parte do programa que desenha os cavalos. Este tipo de GOTO é designado por *incondicional,* porque o salto é realizado sem que seja necessário estar satisfeita qualquer condição especial; daqui a algum tempo discutiremos a forma como o computador toma decisões usando a instrução IF/THEN, entrando então em contacto com os saltos condicionais.

Sugere-se ao leitor que «salte» para o início do Capítulo 11 (se tiver gravador de *cassettes*) ou 12 (se utilizar discos) e leia as primeiras páginas a fim de aprender a gravar programas. Passe o programa «Corrida de Cavalos» para uma *cassette* ou disco, dado que voltaremos a usar este programa mais adiante, e evitará assim voltar a escrevê-lo.

ON... GOTO

Trata-se de uma variante da instrução GOTO, que é usada quando queremos definir uma série de destinos, dependendo do valor de uma determinada variável.

Vejamos um programa simples que mostra o funcionamento de ON... GOTO:

```
10 REM DEMONSTRAÇÃO DE ON GOTO
20 A = INT (RND * 4) + 1
30 FOR B = 1 TO 500 : NEXT
40 ON A GOTO 50, 70, 90, 110
50 PRINT, "UM"
60 GOTO 20
70 PRINT, "DOIS"
80 GOTO 20
90 PRINT, "TRÊS"
100 GOTO 20
110 PRINT, "QUATRO"
120 GOTO 20
```

A linha 20 produz os números um, dois, três e quatro de uma forma aleatória, atribuindo cada valor à variável A. A linha 40, a mais importante para esta nossa discussão, envia a máquina para a linha 50 no caso de A ser igual a um (50 é o primeiro número depois de GOTO), para a linha 70 se A for igual a dois (70 é o segundo número depois de GOTO), para a linha 90 se A for igual a três, e para a linha 110 se A é igual a quatro. Os números de linha 50, 70, 90 e 110 imprimem o número por extenso.

GOSUB e subrotinas

A GOSUB, tal como a GOTO, altera a sequência da execução do programa, mas, em vez de provocar simplesmente um salto para uma linha, a instrução GOSUB produz a passagem a uma «subrotina». Uma subrotina é um bloco de código no interior do programa principal; um programa bem estruturado utilizará muitas subrotinas, todas elas chamadas por uma instrução GOSUB. No final da subrotina deve existir uma instrução RETURN, que reenvia a execução para a instrução que se segue àquela que chamou a subrotina. O programa que se segue torna esta explicação mais clara:

```
10 GOSUB 50 : ' IMPRIMIR TÍTULO
20 GOSUB 90 : ' PEDIR NÚMERO
30 GOSUB 120 : ' IMPRIMIR RESULTADO
40 GOTO 10
50 PRINT : PRINT : PRINT
60 PRINT ''SOMA DE NÚMEROS''
70 PRINT : PRINT
80 RETURN
90 INPUT ''INDIQUE DOIS NÚMEROS A SOMAR'' ;A
100 INPUT B
110 RETURN
120 TOTAL = A + B
130 PRINT ''A RESPOSTA É : ''; TOTAL
140 RETURN
```

O programa principal é apenas formado por três instruções GOSUB e uma instrução GOTO, esta última enviando de novo a execução para o início do próprio programa principal. As três subrotinas no entanto fazem todo o trabalho: a primeira, que se inicia na linha 50, imprime o título; a segunda pede os dois números que se deseja somar, e a última imprime a resposta. Note-se que as instruções REM no final de cada GOSUB são opcionais (dado que o Amstrad ignorará o seu conteúdo), mas servem ao utilizador do programa para saber o que este faz.

A instrução GOSUB pode ser usada com a instrução ON de uma forma semelhante à declaração ON... GOTO, exceptuando o facto de o computador saltar para uma subrotina em vez de uma linha.

Um efeito simples que pode ser obtido usando o processo GOSUB/ /RETURN é uma espécie de poesia. Basta escrever algumas linhas, cada uma numa subrotina separada, criar um número aleatório entre um e o número de linhas «poéticas», e utilizar uma instrução ON... GOSUB que dirija o computador para a linha seleccionada aleatoriamente.

Podemos ver em seguida os resultados de uma súbita inspiração produzida pelas minhas recentes férias na Áustria. O resultado não é espectacular... mas talvez o leitor consiga melhor do que eu!

Eis um extracto dos muitos poemas que o Amstrad compôs para mim:

CIDADE DE SALZBURGO

FORTALEZA INACESSÍVEL CÚPULAS DE COBRE
RUAS EMPEDRADAS
MEMÓRIAS ECOADAS NA PEDRA
FORTALEZA INACESSÍVEL CAMINHOS ESTREITOS
ESQUECIDO ENTRE MÚSICA DE MOZART
FORTALEZA INACESSÍVEL ECOADAS
ENTRE MEMÓRIAS NA PEDRA
SONHANDO
MÚSICA DE MOZART ESPIRAIS BARROCAS
SONHANDO
ESQUECIDO

E agora o programa que permitiu ao Amstrad produzir tão cativantes estrofes:

```
10 REM POESIA USANDO GOSUB
20 WHILE INKEY$=" " : WEND
30 CLS
40 PRINT "CIDADE DE SALZBURGO"
50 PRINT
60 RANDOMIZE TIME
70 P = INT (RND * 22) + 1
80 ON P GOSUB 110,130,150,170,190,210,230,250,270,290,310,
   330,350,370,390,410,430,450,470,490,510,530,550
90 FOR T = 1 TO 200 : NEXT
100 GOTO 70
110 PRINT
120 RETURN
130 PRINT
140 RETURN
150 PRINT
160 RETURN
170 PRINT
180 RETURN
190 PRINT " . . . . . "
200 RETURN
210 PRINT "CÚPULAS DE COBRE"
```

```
220 RETURN
230 PRINT "MÚSICA DE MOZART"
240 RETURN
250 PRINT "DESCANSA SUAVEMENTE";
260 RETURN
270 PRINT "AR DA MONTANHA";
280 RETURN
290 PRINT "RUAS EMPEDRADAS"
300 RETURN
310 PRINT "ESQUECIDO";
320 RETURN
330 PRINT "ESPIRAIS BARROCAS"
340 RETURN
350 PRINT "SONHANDO";
360 RETURN
370 PRINT "ECOADAS";
380 RETURN
390 PRINT "CAMINHOS ESTREITOS";
400 RETURN
410 PRINT "ENSOMBRADO POR";
420 RETURN
430 PRINT "FORTALEZA INACESSÍVEL";
440 RETURN
450 PRINT "ENTRE";
460 RETURN
470 PRINT "CAMINHANDO LENTAMENTE";
480 RETURN
490 PRINT "MEMÓRIAS NA PEDRA";
500 RETURN
510 PRINT "IGREJAS";
520 RETURN
530 PRINT "CÉU PESADO";
540 RETURN
550 PRINT "LÍMPIDA ATMOSFERA";
560 RETURN
```

4

CICLOS

Começaremos agora a estudar alguns comandos que aparecem sempre juntos: FOR e NEXT, WHILE e WEND.

Essencialmente, FOR e NEXT controlam um ciclo que é executado o número de vezes especificado na instrução FOR. O fragmento de programa seguinte deveria esclarecer melhor o que se passa:

```
10 REM FOR / NEXT
20 FOR A = 1 TO 10
30 PRINT ''EIS O NÚMERO '';A;''DO CICLO''
40 NEXT A
```

Quando se executa o programa, este produz o seguinte resultado:

```
EIS O NÚMERO 1 DO CICLO
EIS O NÚMERO 2 DO CICLO
EIS O NÚMERO 3 DO CICLO
EIS O NÚMERO 4 DO CICLO
EIS O NÚMERO 5 DO CICLO
EIS O NÚMERO 6 DO CICLO
EIS O NÚMERO 7 DO CICLO
EIS O NÚMERO 8 DO CICLO
EIS O NÚMERO 9 DO CICLO
EIS O NÚMERO 10 DO CICLO
```

As duas frases EIS O NÚMERO e DO CICLO são impressas dez vezes, sendo o valor actual de A impresso entre ambas. Depois de ter executado o programa, altere a linha 40 para

```
40 NEXT
```

Terá, portanto, eliminado o nome da variável de controlo A; o seu Amstrad continua a compreender a instrução. Mesmo que utilize vários

ciclos FOR/NEXT, o computador saberá ainda como deve acasalar os FOR com os NEXT, desde que exista um número igual de ambas as instruções no programa. Nestas condições, o leitor poderá preferir deixar a variável de controlo no final da instrução NEXT, para facilitar a leitura do programa.

Podemos incluir ciclos uns dentro dos outros em caso de necessidade. Vejamos um exemplo de um ciclo embebido noutro: o ciclo B encontra-se no interior do ciclo A. Esta rotina imprime as tabelas de multiplicação desde 1 × 1 até 12 × 12:

```
10 REM CICLOS EMBEBIDOS
20 FOR A = 1 TO 12
30 FOR B = 1 TO 12
40 PRINT A; "VEZES ";B;" = ";A*B
50 NEXT B
60 NEXT A
```

É importante garantir que a última instrução NEXT que ocorre no programa corresponde à mesma variável de controlo (a letra A, ou qualquer outra) usada no primeiro FOR. Como podemos verificar no programa anterior, a primeira instrução FOR refere-se à variável de controlo A, tal como o último NEXT. A variável de controlo de ciclo mencionada em seguida é a B, e o penúltimo NEXT refere-se geralmente a ela. É vital que as instruções FOR e NEXT estejam dispostas deste modo — se não, o seu computador ficará provavelmente sem saber o que fazer. Pode-se demonstrar isto alterando as linhas 50 e 60 para o seguinte:

```
50 NEXT A
60 NEXT B
```

Isto produzirá uma mensagem de erro. Um modo de resolver o problema consiste em retirar as variáveis de controlo das instruções NEXT, deixando o computador resolver o assunto do melhor modo. Nesse caso, as duas últimas linhas do programa terão a seguinte aparência:

```
50 NEXT
60 NEXT
```

Uma outra maneira de proceder consiste em eliminar a linha 60 (o que se faz escrevendo 60 sem mais nada à frente, e carregando em ENTER) e alterar a linha 50 do seguinte modo:

```
50 NEXT B,A
```

Como seria de esperar, o programa trabalha exactamente do mesmo modo em ambos os casos, só não o fazendo quando as variáveis de controlo são indicadas pela ordem errada. O leitor verificará que o Amstrad funciona ligeiramente mais depressa quando não é forçado a verificar se a variável de controlo está certa. Portanto, o uso de NEXT isoladamente, sem qualquer variável de controlo, constitui provavelmente o melhor modo de executar os ciclos FOR/NEXT. No entanto, convém, no início da sua actividade como programador, usar a versão apresentada no primeiro programa, usando NEXT B e NEXT A. Isto ajudá-lo-á a seguir o funcionamento do programa.

FOR/NEXT... STEP

Existe um outro aspecto dos ciclos FOR/NEXT que devemos discutir imediatamente, a saber, o uso de STEP. Limpe a memória do computador escrevendo NEW, carregando em seguida na tecla ENTER, e escreva o programa:

```
10 REM FOR/NEXT . . . STEP
20 FOR A = 1 TO 20
30 PRINT A
40 NEXT A
```

Quando se executa este programa, o computador imprime os números 1 a 20 no visor. Alteremos agora a linha 20 de modo a ficar assim:

```
20 FOR A = 1 TO 20 STEP 2
```

Quando se executa novamente o programa, o computador imprime apenas os números 1, 3, 5, 7, 9, etc., saltando de dois em dois e terminando em 19. De facto, o programa começou a contar a partir do primeiro número indicado (A = 1) até atingir o número mais próximo possível do último (20), mas contando dois a dois. Altere uma vez mais a linha 20 e volte a executar o programa:

```
20 FOR A = 1 TO 20 STEP 3
```

Desta vez o computador imprimirá no visor os números 1, 4, 7, 10, 13, 16 e 19.

Aprendemos, portanto, que os ciclos FOR/NEXT contam de um em um, a não ser que seja especificado o «passo» (STEP) da contagem.

O computador também pode contar para menos. Não há nada que nos impeça de executar o ciclo FOR/NEXT de mais para menos. Experimente o seguinte:

```
20 FOR A = 20 TO 1 STEP -1
```

Por outro lado, o «passo» não necessita de ser um número inteiro, como podemos descobrir alterando de novo a linha 20 para:

```
20 FOR A = 20 TO 1 STEP -0.7
```

É chegado o momento de voltarmos ao nosso programa «Corrida de Cavalos», apresentado no capítulo anterior, e de estudarmos os ciclos FOR/NEXT nele existentes. Carregue o programa de novo no computador, a partir da *cassette* ou do disco, ou consulte a listagem dada no Capítulo 3.

O primeiro ciclo, um ciclo "T", inicia-se na linha 70 e termina na linha 250.

A primeira acção no interior deste ciclo consiste em subtrair cinco dólares de cada um dos jogadores, aceitando em seguida as apostas dos jogadores. A linha 240 rejeita o cavalo escolhido pelo segundo jogador se for o mesmo do primeiro.

Um dos ciclos FOR/NEXT que se seguem é o da linha 370. Note-se que utiliza igualmente uma variável de controlo T. É perfeitamente aceitável usar a mesma letra para diferentes ciclos no interior do mesmo programa, desde que os ciclos não estejam embebidos uns nos outros. Como os cavalos com os números menores são um pouco favorecidos (dado que são verificados por ordem numérica, a fim de determinar se passaram a linha de chegada, pode acontecer que o cavalo número um ganhe a corrida por ser verificado primeiro), o ciclo da linha 370 dá aos cavalos com números superiores um pequeno avanço no início da corrida.

A corrida propriamente dita é tratada pelo ciclo que usa a variável de controlo B, entre a linha 380 e a 430. Este ciclo é executado repetidamente, acrescentando um pouco (na linha 390) a cada cavalo até um deles (ver a linha 420) possuir um total superior a .32, momento em que a acção é transferida para a linha 470, onde é proclamado o vencedor. As linhas 510 e 520 verificam se qualquer dos jogadores escolheu o cavalo vencedor e, se tal aconteceu, somam 10 dólares à sua conta. A linha 530 utiliza outro ciclo FOR/NEXT para atrasar um pouco o programa, o mesmo acontecendo depois, ao imprimir o texto indicador do início de uma nova corrida. Um outro ciclo de atraso, na linha 600, pára momentaneamente o programa e depois a acção é passada para a linha 60, iniciando-se de novo a corrida.

WHILE e WEND

Estas duas instruções são bastante semelhantes a FOR e a NEXT na medida em que controlam um ciclo... mas de uma forma ligeiramente diferente. No início de um ciclo WHILE/WEND encontra-se a ordem WHILE e depois uma condição. No final do ciclo está a instrução WEND. Se a condição que se segue a WHILE for verdadeira, o programa continua a executar o ciclo; no entanto, depois de a condição se tornar falsa, o programa passa para além da instrução WEND que se encontra no final do ciclo.

Vejamos um exemplo do uso de WHILE/WEND:

```
10 REM DEMONSTRAÇÃO WHILE/WEND
20 TOTAL = 0
30 WHILE TOTAL< >20
40 TOTAL = TOTAL + 1
50 PRINT TOTAL
60 WEND
```

Note-se que o computador continua a somar um à variável T, até T ser igual a 20 e o ciclo ser terminado.

WHILE/WEND pode ser muito útil — uma utilização simples, mas eficaz, consiste em deixar o computador fazer uma pausa na execução do programa até se tocar numa tecla; a instrução para isto é WHILE INKEY$ = " ":WEND. Portanto, enquanto não se carregar em nenhuma tecla (enquanto o comando INKEY$ é ainda igual a uma cadeia nula), o computador executa um ciclo sem fim.

Código

Vejamos um uso um pouco mais complexo de WHILE e WEND:

```
10 'CÓDIGO
20 '
30 INPUT "INDIQUE O SEU CÓDIGO DE UM CARACTER";
   CODE$
40 CLS
50 PRINT "Amstrad 64K Microcomputador (v1)"
60 PRINT
70 PRINT " "+CHR$(164)+"1984 Amstrad Consumer Electronics
   plc and Locomotive Software Ltd."
80 PRINT : PRINT " BASIC 1.0"
90 PRINT : PRINT "Ready"
100 PRINT CHR$(143)
```

```
110 WHILE A$ < >CODE$
120 A$ = INKEY$
130 WEND
140 SOUND 1,300,40
150 PRINT "ESTÁ AGORA NO SISTEMA CPC 464"
160 END
```

Neste programa escrevemos um único caracter que servirá como código pessoal; o caracter em causa pode ser um caracter de controlo. A Amstrad imprime no visor exactamente aquilo que habitualmente imprime ao ser ligado. Até o utilizador carregar na tecla correcta, não consegue usar o computador — a menos que carregue na tecla de BREAK, que pára a execução do programa. No Capítulo 22, sobre os modos de programar mais profissionalmente, discutiremos as instruções necessárias para desactivar a tecla BREAK.

A condição WHILE ocorre quando a entrada do utilizador não é igual ao código do caracter que o programa está preparado para aceitar; quando se toca na tecla correcta, o computador abandona o ciclo.

Note-se que na última linha, linha 160, se utilizou o comando END. Esta instrução, como se pode imaginar (dado que significa «fim»), termina o programa e é semelhante à instrução STOP, exceptuando o facto de a STOP provocar um BREAK do programa a partir do qual é possível continuar usando a instrução CONT. Substitua a instrução END por STOP na linha 160, execute o programa e veja a diferença entre as duas instruções finais.

Um último aspecto. Ao brincar um pouco com o Amstrad descobrimos CALL &BB18, uma instrução que simula a série de comandos WHILE INKEY$=" ":WEND. A máquina não recorda a tecla carregada, mas se o leitor quiser alguma vez recorrer a uma linha de pausa do tipo «carregue em qualquer tecla», como tanta vez se utiliza em programas, a forma mais simples e rápida de a obter consiste em recorrer ao uso desta instrução CALL. O leitor deverá porém ter bastante cuidado ao usá-la, não a confundindo com a instrução semelhante CALL BB18. Com efeito, esta última instrução parece provocar um *reset* do sistema; por outras palavras, o programa que estiver a executar desaparece e o computador fica exactamente como se tivesse acabado de ser ligado.

5

CONTINUANDO A APRENDER

Vejamos o que aprendemos até agora. Já avançámos muito; o leitor deve já conhecer as palavras que se seguem, sendo mesmo capaz de as manipular com alguma facilidade:

PRINT	LIST	NEW
RUN	TAB	CLS
REM	GOSUB	CONT
FOR/NEXT	ON... GOTO	ON... GOSUB
WHILE	WEND	FOR/NEXT...STEP
PEN	PAPER	INK
END	STOP	

Além destas palavras, o leitor aprendeu o que são as linhas de programa, as variáveis numéricas e de cadeia e o modo de usar o ponto e vírgula nas instruções PRINT.

Já conseguimos portanto alguma coisa e, de vários pontos de vista, o pior já passou. A «curva de aprendizagem» da linguagem BASIC sobe lentamente, sendo a parte mais inclinada a que corresponde às primeiras horas de aprendizagem. Depois de passarmos essas horas — o que já aconteceu neste momento —, descobre-se que tudo o que se aprende de novo se liga facilmente àquilo que já se domina. Isto transforma a aprendizagem num processo agradável e não muito exigente.

Linhas com várias instruções

Usámos linhas deste tipo em alguns dos programas anteriores. Uma linha com várias instruções tem a seguinte aparência:

```
10 FOR T = 1 TO 20 : PRINT T : PRINT T/2 : NEXT T
```

Como o leitor descobrirá se executar esta linha, trata-se de facto de um programa completo, equivalente às quatro linhas seguintes:

```
10 FOR T = 1 TO 20
20 PRINT T
30 PRINT T/2
40 NEXT T
```

Se executar este programa de quatro linhas, verificará que produz exactamente o mesmo resultado que o anterior.

Em geral, convirá ao leitor evitar as linhas com várias instruções, dado que tornam os erros mais difíceis de detectar e podem tornar muito difícil a procura de erros no programa. Também se pode cair em algumas ratoeiras. Por exemplo, nas duas linhas que se seguem, a segunda instrução de qualquer delas nunca será executada, apesar de estar separada da primeira da forma habitual (usando dois pontos):

```
10 GOTO 50 : A = 25
10 REM FIM DO PROGRAMA : PRINT "ADEUS"
```

O uso de linhas de uma ou várias instruções depende em grande medida do gosto do programador.

Em certos momentos as linhas de várias instruções são desejáveis, e mesmo essenciais. Por exemplo, são desejáveis quando se introduz um espaçamento entre linhas de impressão:

```
10 PRINT ''ÁGUIAS''
20 PRINT : PRINT
30 PRINT ''EM VOO''
40 PRINT : PRINT
50 GOTO 10
```

Pode-se igualmente recorrer a elas quando se usa um ciclo FOR/ /NEXT «falso» para introduzir um atraso:

```
10 PRINT ''ISTO NUNCA MAIS ACABA''
20 FOR T = 1 TO 1500 : NEXT T
30 END
```

Algumas funções

O leitor já verificou que os programas estudados até agora incluem muitas instruções BASIC ainda não explicadas. Isto deve-se ao facto de, até certo ponto, se tornar cada vez mais difícil evitar programas

sem palavras desconhecidas, em especial se se deseja que tais programas tenham alguma utilidade. Algumas das instruções que o leitor ainda não conhece têm uma base matemática e serão familiares a quem tiver alguns conhecimentos neste campo. Outras palavras não serão tão familiares, mas todas elas merecem alguns minutos de atenção.

Funções matemáticas

ATN

Esta função produz o arco com uma dada tangente e é usada na seguinte forma: A = ATN(n). Dá uma resposta em radianos, e funciona no domínio PI/2 a −PI/2.

COS

Esta instrução produz o co-seno e é usada do seguinte modo: C = COS(n). Dá-nos uma resposta em radianos.

DEG

Esta função pode ser usada isoladamente, como em 20 DEG; isto passa qualquer cálculo que envolva SIN e COS (como o desenho de um círculo) para o modo «graus», em vez do modo «radianos».

EXP

Esta função está ligada à instrução LOG, dado que transforma um logaritmo (criado usando LOG) no seu valor original. Por exemplo:

```
PRINT LOG(10)    2.30258509
PRINT EXP(2.30258509)    9.99999996
```

LOG

Esta função produz o logaritmo natural de um número e é usada na forma A = LOG(n)

LOG 10

A função LOG 10 produz o logaritmo de um número na base 10.

PI

São poucas as pessoas que não ouviram já falar em PI, correspondente a um valor de 3,14159265 (pelo menos o nosso Amstrad só chega aqui). É muito usado para diversos cálculos envolvendo círculos e linhas curvas.

RAD

A função RAD passa os cálculos para o modo «radianos». Note-se que o modo automático de funcionamento do computador é em radianos.

SIN

A função SIN fornece o seno de um ângulo e possui o seguinte formato: S = SIN(n). A resposta é igualmente dada em radianos.

SQR

SQR (*square root*, raiz quadrada) calcula obviamente a raiz quadrada de um número.

TAN

Esta função produz a tangente de um número, sendo usada do seguinte modo: T = TAN(n). Esta função produz igualmente o seu resultado em radianos.

Funções «figurativas»

O Amstrad possui várias funções que actuam sobre um valor, cortando o número dos seus algarismos ou arredondando-o para um valor específico.

ABS

ABS calcula o valor absoluto de um número, ou seja, aceita o número mas não tem em conta o seu sinal. Isto pode ser muito útil quando se determina a diferença entre duas variáveis.

CINT

A função CINT converte números reais em inteiros, mas arredonda para o valor mais próximo. Assim, CINT(2.7) dará o valor 2. Note-se que .5 de um valor (como em 3.5) será arredondado para um número inteiro. O leitor deverá não esquecer que os computadores que «falam» inglês usam ponto decimal e não vírgula decimal ao representar valores fraccionários.

CREAL

Esta instrução cria um número real a partir de um inteiro.

FIX

Esta função guarda apenas a parte do número que se encontra à esquerda do ponto (vírgula) decimal, pelo que FIX(74.9327) produz 74. De facto, FIX é bastante semelhante a INT.

INT

Provavelmente a função mais usada, INT produz um inteiro a partir de um valor «real» — isto é, de um número com parte fraccionária. Assim, INT(203.6) produz 203. Note-se que INT arredonda sempre para menos.

MAX

Se colocarmos entre parêntesis uma lista de expressões numéricas, a função MAX escolhe a de maior valor. Portanto, se tivermos a instrução PRINT MAX (23, 47, 6, 9, 8, 107, 56, 88), o resultado será 107.

MIN

A função MIN é exactamente oposta a MAX, na medida em que determina o número mais pequeno de uma lista. Note-se que em ambas as funções se podem utilizar variáveis em vez de números. Por exemplo, MIN (A, B, C, 107, D), onde A = 90, B = 109, C = 34 e D = 77, escolhe C.

ROUND

A função ROUND arredonda um número da mesma maneira que CINT, mas pode-se especificar o número de casas decimais pretendidas no resultado. Portanto, ROUND (2.38769,3) produz o resultado 2.388 (devido ao três que se segue à vírgula).

SGN

Esta função determina o sinal de uma variável. Se esta for maior do que zero, produz 1; e se for menor do que zero, −1. Se a variável tiver um valor nulo, a função produz 0.

UNT

Esta função converte um inteiro de 16 bits sem sinal, como &BB18, num inteiro de 8 bits na gama entre −32768 e 32768. A sua principal utilidade ocorre na conversão de endereços de 16 para 8 bits. Assim, uma instrução como CALL &BB18 pode também assumir a forma CALL −17640, dado que é este o valor obtido quando se escreve PRINT UNT (&BB18).

Vejamos um pequeno programa de demonstração que ilustra o funcionamento destas funções:

```
10 ' DEMONSTRAÇÃO DAS FUNÇÕES
20 A = −10
30 B = 7.894325
40 C = 3498.5
50 D = 29.3786
60 CLS
70 PRINT TAB(15); ''FUNÇÕES.''
80 PRINT
90 PRINT ''O MAIOR VALOR É ''; MAX (A,B,C,D)
100 PRINT ''O MENOR VALOR É ''; MIN(A,B,C,D)
120 PRINT
130 PRINT ''OS VALORES ABSOLUTOS SÃO'' : PRINT ABS(A);
    '';ABS(B);'' '';ABS(C);'' '';ABS(D)
140 PRINT
150 PRINT ''OS SINAIS SÃO''; SGN(A); SGN(B); SGN(C); SGN(D)
160 PRINT
170 PRINT ''CINT DE A,B,C,D'': PRINT CINT(A); CINT(B);
    CINT(C); CINT(D)
180 PRINT
```

```
190 PRINT "INT DE A,B,C,D" : PRINT INT(A); INT(B); INT(C);
    INT(D)
200 PRINT
210 PRINT "FIX DE A,B,C,D": PRINT FIX(A); FIX(B); FIX(C);
    FIX(D)
220 PRINT
230 PRINT "ARREDONDAMENTO PARA DUAS CASAS DECI-
    MAIS": PRINT ROUND(A,2); ROUND(B,2); ROUND(C,2);
    ROUND(D,2)
240 END
```

Definição de funções pelo utilizador

A ordem DEF FN permite-nos definir as nossas próprias funções. Esta instrução pode ser muito útil no caso de um cálculo complicado que deva ser executado várias vezes no mesmo programa. A sintaxe da ordem DEF FN é:

DEF FN nome (parâmetros formais) = expressão numérica.

No programa que se segue podemos encontrar um exemplo da definição de uma função. O programa calcula o pagamento de um empréstimo:

```
10 ' DEFINIÇÃO DE FUNÇÕES
20 '
30 DEF FN TOTAL (P) = ((P*I)/100) + P
40 INPUT "JURO";I
50 INPUT "EMPRÉSTIMO";P
60 PRINT "TOTAL A PAGAR"; FN TOTAL(P)
```

As três instruções DEF INT, DEF REAL e DEF STR$ constituem formas diferentes de definição de funções. A primeira, DEF INT, permite-nos definir certas variáveis como inteiros. Por agora, todas as nossas variáveis numéricas, A, ZZT, PONTUAÇÃO, etc., são números reais — por outras palavras, podem ter casas decimais. Muitas das variáveis que utilizamos só necessitam porém de ser inteiras, e para o computador o cálculo e manipulação de variáveis inteiras é mais rápido e eficiente.

Outro modo de criar variáveis inteiras consiste em escrever depois do nome da variável um sinal de percentagem. No entanto, isto deverá ser feito ao longo de todo o programa. É de facto muito mais simples definir as variáveis como inteiras no início do programa.

A sintaxe deste comando é DEF INT seguido dos nomes das variáveis separadas por vírgulas. Se quisermos usar todo um conjunto de

letras como nomes de variáveis inteiras, em vez de escrevermos todas, podemos recorrer a uma ordem com o formato DEF INT A – X. Todas as variáveis cujo nome é uma letra entre A e X, inclusive, são consideradas inteiras.

DEF REAL funciona exactamente do mesmo modo que DEF INT, mas considerando as variáveis listadas depois de DEF REAL como números reais. Finalmente, DEF STR$ converte variáveis numéricas em variáveis de cadeia.

Convém referir alguns pontos antes de abandonarmos este problema da definição de funções. Em primeiro lugar, e se bem que no texto tenhamos escrito as duas partes da função de definição separadas por um espaço, fizemo-lo apenas para facilitar a leitura — não use o espaço nos seus programas. Em segundo lugar, nenhuma destas instruções pode ser usada em modo directo, fora de um programa; devem ser usadas apenas em programas.

6

READ/DATA E RESTORE

Trata-se de um conjunto de palavras muito úteis, que geralmente são empregues juntamente em cada programa. Basicamente, a instrução READ é usada para aceder informações guardadas em declarações DATA, e RESTORE é usada para indicar onde — numa série de declarações DATA — desejamos começar a ler a informação.

O programa que se segue deverá tornar mais claro o uso destas instruções:

```
10 REM DEMONSTRAÇÃO READ/DATA
20 FOR A = 1 TO 10
30 READ B
40 PRINT B
50 NEXT A
60 RESTORE
70 FOR C = 1 TO 800 : NEXT C
80 PRINT
90 FOR A = 1 TO 10
100 READ B
110 PRINT B
120 NEXT A
130 DATA 12,76,85,333,9,23,43,65,88,2893
```

Quando se executa este programa, vê-se que os números da declaração DATA, na linha 130, são impressos um a um, à medida que o computador executa o primeiro ciclo A, e repetidos quando executa o segundo ciclo A.

Quando se executa um programa, o computador executa automaticamente uma ordem RESTORE para a primeira linha de DATA (Dados) existente naquele. Quando o Amstrad encontra uma instrução RESTORE que não é acompanhada por qualquer número de linha, executa de novo essa instrução para a primeira linha de dados existente no programa.

Vejamos um segundo programa que mostra mais claramente o funcionamento de RESTORE:

```
10 FOR A = 1 TO 10
20 READ B
30 PRINT B
40 IF A = 5 THEN RESTORE 70
50 FOR C = 1 TO 50 : NEXT C
60 DATA 1,2,3,4,5,6,7,8
70 DATA 11,12,13,14,15,16,17,18
```

Quando se executa este programa, verificamos que imprime os números 1, 2, 3, 4, 5, 11, 12, 13, 14 e 15. A instrução RESTORE passou o indicador da posição dos dados para o início da linha 70, e o computador começou a ler dados a partir desta linha a seguir a imprimir o 5 (note-se que não interessa o número de dados que se possui, desde que se tenha pelo menos tantos como aqueles que serão lidos).

Há um aspecto destas instruções que o leitor talvez considere um pouco surpreendente. Não interessa qual é o ponto do programa onde os dados se encontram — o computador encontrará sempre os dados, e lê-los-á. Vejamos, por exemplo, o programa seguinte:

```
10 DATA 1,2
20 FOR A = 1 TO 11
30 DATA 3,4,5
40 READ B
50 DATA 6,7
60 PRINT B
70 DATA 8,9,10
80 NEXT A
90 DATA 11
```

Até este momento temos lido dados numéricos e comparado variáveis com números existentes em declarações DATA. Passemos agora às cadeias alfanuméricas, por exemplo com o programa seguinte:

```
10 FOR A = 1 TO 5
20 READ H$
30 PRINT H$
40 NEXT
50 DATA "OLÁ", "COMO", "ESTÁS", "MÁQUINA",
"ESPERTA"
```

Todas as palavras foram encerradas dentro de aspas, mas estas não são estritamente necessárias. Apenas necessitamos delas quando queremos incluir pontuação na declaração de dados, por exemplo "OLÁ, COMO".

A informação numérica e de cadeias pode ser misturada dentro de um programa, desde que se garanta que o computador encontra um número quando está à procura dele, e uma cadeia quando espera uma. O programa que se segue, por exemplo, consegue obter a informação apropriada no momento certo:

```
10 FOR A = 1 TO 5
20 READ H$,B
30 PRINT H$,B
40 NEXT
50 DATA ESTA,1,DEMONSTRAÇÃO,2,É,3,BASTANTE,4,CURTA,5
```

O resultado da execução será:

```
ESTA            1
DEMONSTRAÇÃO    2
É               3
BASTANTE        4
CURTA           5
```

Para terminarmos a nossa discussão de READ/DATA/RESTORE, vejamos um uso mais agradável das declarações de dados. Nesta versão simples de um jogo conhecido, deve-se adivinhar a palavra misteriosa escrevendo uma letra que se pense poder existir nela. Podemos errar sete letras antes de perdermos. Quando conhecemos a solução, escrevemos a palavra inteira e o Amstrad indica-nos se acertámos.

Antes de introduzirmos o programa na máquina, convém referir alguns pontos. Em primeiro lugar, se o leitor quiser utilizar um vocabulário diferente do empregue no programa, basta-lhe alterar o conteúdo das declarações DATA. Por outro lado, note a forte estrutura do programa — voltaremos a este ponto mais tarde. Para já, divirta-se com o programa!

```
10 ' JOGO
20 '
30 GOSUB 370
40 GOSUB 310
50 GOSUB 230
60 IF AT > 6 THEN 100
70 IF G$ = A$ THEN 160
80 AT = AT + 1
90 GOTO 40
100 ' PERDER
110 SOUND 1,2000,200
```

```
120 CLS
130 LOCATE 12,12 : PRINT "PERDEU"
140 PRINT : PRINT " A PALAVRA ERA ";A$
150 END
160 ' GANHAR
170 CLS
180 FOR T = 200 TO 80 STEP – 20
190 SOUND 1,T
200 NEXT
210 LOCATE 10,12 : PRINT "ÓPTIMO, GANHOU!"
220 STOP
230 ' ENTRADAS
240 LOCATE 1,20
250 INPUT "ADIVINHE UMA LETRA (OU PALAVRA INTEIRA
    SE JULGA CONHECÊ-LA)";G$
260 IF G$ = " " THEN 250
270 FOR T = 1 TO LEN(A$)
280 IF MID$(A$,T,1 ) = LEFT$(G$,1) THEN
    MID$(B$,T,1) = MID$(A$,T,1)
290 NEXT
300 RETURN
310 ' IMPRIMIR
320 CLS
330 PRINT : PRINT "JOGO DAS PALAVRAS"
340 PRINT : PRINT
350 PRINT B$
360 RETURN
370 ' ESCOLHER PALAVRA
380 AT = 0
390 RESTORE
400 FOR T = 1 TO INT (RND*15) + 1
410 READ A$
420 NEXT
430 FOR T = 1 TO LEN (A$)
440 B$ = B$ + "—"
450 NEXT
460 RETURN
470 DATA "ALFABETO", "PREFABRICADO", "RIDÍCULO"
480 DATA "MÁQUINA", "INDIVÍDUO", "CONSOLIDAR"
490 DATA "METAFÍSICO", "EMPREENDIMENTO",
    "RELEVANTE"
500 DATA "CÍTARA", "OBOÉ", "SEQUELAS", "TRAPÉZIO",
    "CRIPTA"
510 DATA "TIRANO", "DEMOCRACIA", "COMÉRCIO"
520 DATA "DETERMINADO", "EDITOR"
```

A instrução DIM é usada quando desejamos construir um *array* contendo uma lista de números, que podem depois ser acedidos indicando o elemento do *array* pretendido.

Vejamos um exemplo que esclarecerá melhor esta questão.

No caso de um *array* que guarde números, usamos um *array* da forma DIM A(20) quando pretendemos que o *array* contenha 21 elementos (o número de elementos de um *array* é quase sempre mais um do que o número especificado entre parêntesis na instrução DIM). Cada elemento do *array* é preenchido com zero no momento em que é dimensionado.

O programa que se segue DIMensiona um *array* na linha 10 e depois, usando instruções READ, preenche cada elemento do *array* com um número. A secção final do programa imprime todos os elementos, de uma forma que deverá contribuir para esclarecer o que se passa. Note-se que o primeiro elemento do *array* é o A(0), o segundo A(1), e assim por diante, até A(20).

```
10 DIM A(20)
20 FOR B = 0 TO 20
30 A(B) = INT (RND * 10) + 1
40 NEXT B
50 FOR B = 0 TO 20
60 PRINT "A(";B;") É ";A(B)
70 NEXT B
```

Um *array* é muito útil quando desejamos escolher aleatoriamente entre várias possibilidades... como se mostra no programa seguinte.

```
10 DIM A(5)
20 FOR B = 0 TO 5
30 READ A(B)
40 NEXT B
50 PRINT A (INT (RND * 6))
60 FOR C = 1 TO 300 : NEXT
70 GOTO 50
80 DATA 999,2345,1111,2,9,42
```

Os *arrays* que vimos até agora foram sempre unidimensionados. Um *array* multidimensionado é definido acrescentando à letra identificadora do *array* mais de um índice. Escreva e execute o programa seguinte, e observe as suas saídas:

```
10 DIM A(4,4)
20 FOR B = 0 TO 4
30 FOR C = 0 TO 4
40 A (B,C) = INT (RND * 9)
50 NEXT C,B
60 FOR B = 0 TO 4
70 FOR C = 0 TO 4
80 PRINT B;C;"     ";A(B,C)
90 NEXT C,B
```

A saída do programa será a seguinte:

```
00 > 1
01 > 7
02 > 1
03 > 3
04 > 5
10 > 1
11 > 0
12 > 0
13 > 5
14 > 7
20 > 0
21 > 0
22 > 2
23 > 5
24 > 6
30 > 7
31 > 7
32 > 8
33 > 0
34 > 6
40 > 1
41 > 4
42 > 8
43 > 1
44 > 6
```

Se dimensionarmos o *array* colocando a instrução DIM no início do programa, fica claramente definido o tamanho da variável usada. No entanto, se o *array* tem uma única dimensão e esta for inferior a 10, não há necessidade sequer de o dimensionar; o Amstrad reserva automaticamente memória para todos os *arrays* com menos de 11 elementos.

O Amstrad permite usar também *arrays* de cadeias, com um cifrão a seguir ao nome do *array* na declaração de dimensionamento e em todas as referências subsequentes àquele. No programa que se segue é dimensionado um *array* de cadeias na linha 10, cujos elementos são definidos pelo ciclo das linhas 20 a 40 e seleccionados aleatoriamente pela linha 50 a partir das declarações DATA das linhas 80 e 90.

```
10 DIM A$(5)
20 FOR B = 0 TO 5
30 READ A$(B)
40 NEXT B
50 PRINT A$ (INT (RND * 6))
60 FOR C = 1 TO 250 : NEXT C
70 GOTO 50
80 DATA "SÃO", "ESTAS", "AS"
90 DATA "PALAVRAS","A","MISTURAR"
```

Finalmente, o que acontece se dimensionarmos um grande *array*, gastando bastante memória, e depois descobrirmos que não necessitamos dele? A BASIC do Amstrad possui um comando chamado ERASE que elimina simplesmente o *array*. Para apagar o *array* desnecessário basta escrever o seu nome depois da palavra ERASE. Se quisermos eliminar A$(250) e LT(20,40), basta escrever (como linha de programa) ERASE A$,LT.

7

NÚMEROS ALEATÓRIOS E DECISÕES

Já encontrámos números aleatórios anteriormente (no Capítulo 3). Estes números, como o seu nome sugere, são produzidos aleatoriamente pelo computador. Esta afirmação pode ser um tanto enganadora porque o computador, de facto, escolhe números a partir de uma tabela existente na sua memória. Para evitar que utilize a mesma área da tabela repetidamente, usamos a instrução RANDOMIZE seguida de alguma variável. Para obtermos uma variação máxima, vale a pena recorrer ao relógio interno do computador, o que obrigará a máquina a usar como valor inicial da tabela um número que depende do tempo que o computador já esteve ligado.

Assim, em todos os programas que utilizem números aleatórios deve-se recorrer à instrução RANDOMIZE TIME no início. Para poupar memória, podemos dar esta ordem directamente, sem a incluir no próprio programa. Procedemos deste modo para a maior parte dos programas deste livro e portanto, se não vir a instrução citada no programa, dê-a directamente. Isto só se aplica obviamente aos jogos, assim como aos programas de gráficos e sons incluídos no livro.

Passemos ao gerador de números aleatórios propriamente dito. A instrução que cria estes números é a RND. Utilizada isoladamente, produz números entre zero e um. Escreva o programa de duas linhas que apresentamos em seguida e deixe-o em execução durante alguns segundos. Deverá obter resultados semelhantes aos indicados:

```
10 PRINT RND;
20 GOTO 10
```

```
0.498153843  0.42731833
0.668242901  0.343849098
0.564465723  0.326045161
0.893792276  0.362325066
0.439642927  0.122611851
```

0.672479611 0.7077513
7.18285E-02 0.114154763

Estes números não terão grande utilidade por si sós, os números aleatórios de que mais necessitamos são inteiros. Podemos usar fracções do tipo IF RND > .5 THEN PRINT "VOCÊ ESTÁ MORTO!", o que dá ao jogador uma probabilidade de 50% de sobreviver; e as fracções de números aleatórios são muitas vezes usadas em tomada de decisões, particularmente em jogos e simulações — mas os limites definidos em termos de números inteiros são muito mais fáceis de utilizar (a propósito, não se preocupe com o comando IF/THEN usado no exemplo anterior; trata-se da principal instrução de tomada de decisões do nosso computador, e estudá-la-emos ainda neste capítulo).

Como podemos então obter números aleatórios entre dois limites por nós especificados? Se quisermos simular a pontuação de um jogo de futebol, definiremos os limites para cada equipa entre zero e cerca de sete (para obtermos uma resposta realista). Teremos então de fazer duas coisas: primeiro, teremos de levar o gerador de números aleatórios a produzir números entre zero e sete (o que se consegue multiplicando o valor RND por sete) e, em segundo lugar, devemos converter os números resultantes em inteiros — o que é feito utilizando a função INT. No entanto, a INT arredonda as fracções para baixo, pelo que a probabilidade de obter um sete como número aleatório são nulas. Teremos, portanto de trabalhar com um factor correctivo mais elevado, oito em vez de sete. O formato final da instrução RND será

```
PRINT INT (RND * 8); INT (RND * 8)
```

Suponhamos que queremos agora alterar o gerador de números aleatórios de modo a garantir que ambas as equipas conseguem obter pelo menos um golo. Teremos então de conseguir um número aleatório compreendido entre um e sete. Basta-nos somar um ao número aleatório, e diminuir o factor correctivo. A instrução passará a ter a seguinte aparência:

```
PRINT INT (RND * 7) + 1; INT ; (RND * 7) + 1
```

Uma regra fácil de recordar é que o número somado à instrução RND corresponde ao limite inferior dos números a obter. O número que é multiplicado pela função RND é o limite superior.

Lançamento de dados

Os números aleatórios podem ser usados para simular acontecimentos casuais da vida quotidiana, como o resultado do lançamento de uma moeda um certo número de vezes. Os dados produzem ao ser lançados, se não estiverem falseados, números aleatórios entre um e seis, e é muito fácil levar o computador a produzir o mesmo resultado. Execute o programa que se segue:

```
10 A = INT (RND * 6) + 1
20 CLS
30 ON A GOSUB 60,80,100,120,140,160
40 WHILE INKEY$ = " ":WEND
50 GOTO 10
60 PRINT : PRINT "@"
70 RETURN
80 PRINT "@" : PRINT : PRINT "@"
90 RETURN
100 PRINT "@" : PRINT "@" : PRINT "@"
110 RETURN
120 PRINT "@ @" : PRINT : PRINT "@ @"
130 RETURN
140 PRINT "@ @" : PRINT "@" : PRINT "@ @"
150 RETURN
160 PRINT "@ @" : PRINT "@ @" : PRINT "@ @"
170 RETURN
```

Muitos programas de dados exigem que se lancem os dados duas vezes, somando depois os resultados. À primeira vista, poderemos pensar que a maneira de conseguir o mesmo resultado com o computador consistiria em obter números aleatórios entre 2 e 12 de uma só vez. No entanto, não é assim. Se bem que a probabilidade de um dado produzir qualquer resultado seja 1 em 6 (cerca de 17%), a distribuição dos totais produzidos por dois dados é a seguinte: total 2 (2,77%), total 3 (5,55%), total 4 (8,33%), total 5 (11,11%), total 6 (13,88%), total 7 (16,66%), total 8 (13,88%), total 9 (11,11%), total 10 (8,33%), total 11 (5,55%) e total 12 (2,77%).

Os números diferentes devem-se às diversas maneiras de poder ser atingido cada total. Sete, por exemplo, pode ser obtido de seis maneiras diferentes (1 + 6, 2 + 5, 4 + 3, 3 + 4, 5 + 2 e 6 + 1), enquanto um total de dois ou doze só pode ser atingido de uma maneira (1 + 1 ou 6 + 6).

Se um gerador de números aleatórios estiver a funcionar correctamente, e for usado para demonstrar os resultados possíveis de dois

dados, seria de esperar que a distribuição dos totais se aproximasse da distribuição teórica dada no parágrafo precedente. Experimentemos para ver. O programa que se segue simula o lançamento de dois dados, seguido do resultado. Note-se que neste caso os dados são lançados 100 vezes.

```
10 REM LANÇAMENTO DE DADOS
20 DIM A(12)
30 FOR B = 1 TO 100
40 C = INT (RND * 6) + 1
50 C = C + INT (RND * 6) + 1
60 A(C) = A(C) + 1
70 NEXT
80 FOR B = 1 TO 12
90 PRINT B; ''—'';A(B);''%''
100 NEXT
```

```
 1 — 0 %
 2 — 4 %
 3 — 7 %
 4 — 9 %
 5 — 4 %
 6 — 15 %
 7 — 20 %
 8 — 13 %
 9 — 11 %
10 — 9 %
11 — 6 %
12 — 2 %
```

Como se pode verificar, a distribuição de totais aproxima-se bastante da já indicada, dando em particular uma probabilidade maior à volta do número sete.

O programa foi modificado para lançar os dados 1000 vezes, pensando que os resultados se aproximariam mais dos previstos. Vejamos o resultado do lançamento de um par de dados 1000 vezes:

```
1 — 0 %
2 — 4 %
3 — 6 %
4 — 10 %
5 — 10 %
6 — 12 %
```

7 — 15 %
8 — 15 %
9 — 11 %
10 — 7 %
11 — 6 %
12 — 3 %

Como se pode verificar, o resultado é de facto mais semelhante à distribuição teórica.

Decisões, decisões

Como a instrução IF/THEN já foi incluída em alguns exemplos, talvez o leitor esteja suficientemente curioso para a estudar.

A declaração IF/THEN é uma tomada de decisões condicional — SE (if) a condição é satisfeita, ENTÃO (then) fazer alguma coisa. Consideremos, por exemplo, a linha de programa IF A = 5 THEN GOTO 200. Isto significa «se a variável A for igual a 5, saltar para a linha 200 do programa»; chamar-se-á a isto um salto *condicional*, oposto ao salto incondicional já discutido no Capítulo 3.

IF/THEN é uma instrução que permite ao computador actuar como se pensasse e fosse capaz de realizar acções racionais baseadas em decisões próprias. Passemos agora aos comandos que podem ser usados com a declaração IF/THEN. Em primeiro lugar, vejamos as comparações que o computador pode realizar, usando as três instruções AND, OR e NOT.

Comparações

O Amstrad pode usar instruções como AND, OR e NOT para comparar duas ou mais coisas, e pode actuar de modo diferente em função do resultado dessas comparações.

AND

Se as duas instruções a comparar estiverem ligadas pela função AND, o Amstrad só actuará se ambas forem verdadeiras.

```
10 A = 100
20 B = 200
30 IF A = 100 e B = 200 THEN PRINT "SIM"
```

OR

Se as duas instruções estiverem ligadas pela função OR, o resultado será verdadeiro se qualquer delas, ou ambas, o forem.

```
10 A = 100
20 B = 200
30 IF A = 100 OR B = 100 THEN PRINT "SIM"
```

AND e OR podem ser usadas na mesma linha de programa, como se mostra no exemplo seguinte. Neste programa, o Amstrad indicará "SIM" se A e B forem maiores do que 10 ou A igual a B. Experimente com um número de valores de amostra para A e B, até ter a certeza de que compreende o que se passa.

```
10 INPUT "A";A
20 INPUT "B";B
30 IF A > 10 AND B > 10 OR A = B THEN PRINT "SIM"
40 GOTO 10
```

NOT

Esta função é usada para verificar uma condição pela negativa, como se pode ver na breve rotina que se segue:

```
10 INPUT "A";A
20 INPUT "B";B
30 IF NOT A = B THEN PRINT "NÃO"
40 GOTO 10
```

A parte final deste capítulo tem a ver com outro acrescento à IF/THEN — a ELSE. Esta permite que seja executada uma de duas acções em função do resultado do teste da condição. Se a condição é preenchida, obtém-se um resultado; SE NÃO (else), obtém-se outro.

A instrução IF/THEN... ELSE permite diminuir o espaço e o tempo necessários para a tomada de decisão pelo computador. Constitui um modo simples de controlar a tomada de decisões, desde que as próprias condições não sejam muito complexas. Não esqueçamos que podemos usar mais do que uma acção depois de uma THEN ou uma ELSE, bastando usar linhas de várias instruções. Portanto, a instrução IF A = 15 THEN PRINT A : GOTO 120 ELSE PRINT B : END é válida.

8

TRATAMENTO DE CADEIAS

O leitor recordará, da nossa discussão sobre variáveis, que as cadeias são designadas por nomes de variáveis que terminam por um cifrão. As cadeias são partes muito úteis do nosso vocabulário BASIC, dado que podem ser manipuladas de modo a produzirem um certo número de resultados válidos.

As funções de cadeias que iremos observar nesta secção serão STR$, LEN, VAL, LEFT$, RIGHT$ e MID$. Se bem que uma expressão contendo algumas destas instruções possa parecer pouco clara, são fáceis de usar desde que se tenha algum cuidado.

STR$

Esta instrução transforma um número na cadeia equivalente. Ou seja, passa 45.6 a "45.6" — uma transformação que não parece imediatamente ter muita utilidade. No entanto, é útil em algumas circunstâncias porque as cadeias são por vezes mais fáceis de manipular do que os números. Uma das possíveis utilizações será ilustrada adiante, mas primeiro devemos observar outras funções de cadeia.

LEN

Esta função determina o comprimento de uma cadeia. Assim, se dissermos ao computador que deve imprimir LEN (A$), onde A$ é "FULTON", produzirá 6. Se A$ for "45.6", LEN(A$) será 4, dado que o ponto decimal conta como um dos caracteres da cadeia.

VAL

Esta instrução actua ao contrário do STR$; ou seja, STR$(42.6) é "42.6", enquanto VAL("42.6") é 42.6.

Podemos usar duas destas instruções para produzir um programa que alinhe números pela direita, em vez da esquerda, ao contrário do que é habitual:

```
10 FOR J = 1 TO 5
20 READ A
30 A$ = STR$(A)
40 PRINT TAB (10 - LEN(A$));A$
50 NEXT
60 DATA 23.45,350.07,1.34,1444.75,23.46
```

Vejamos as saídas obtidas:

```
  23.45
 350.07
   1.34
1444.75
  23.46
```

A linha 20 lê o número da declaração DATA e a linha 30 passa este número (A) para cadeia (A$). A linha 40 imprime esta, usando a declaração TAB de 10 menos o comprimento (LEN) da cadeia.

LEFT$, RIGHT$ e MID$

Estas instruções são usadas para extrair porções de cadeia: a LEFT$ produz uma porção do lado esquerdo da cadeia, a RIGHT$ uma porção do seu lado direito e a MID$ uma parte da cadeia, com um dado comprimento, a meio daquela.

Vejamos um programa que mostre o uso de LEFT$, RIGHT$ e MID$:

```
10 A$ = "DAKOTA"
20 PRINT "LEFT$(A$,3) = ";LEFT$(A$,3)
30 PRINT "RIGHT$(A$,3) = ";RIGHT$(A$,3)
40 PRINT "MID$(A$,2,3) = ";MID$(A$,2,3)
```

Resultados:

```
LEFT$(A$,3) = DAK
RIGHT$(A$,3) = OTA
MID$(A$,2,3) = AKO
```

Como se pode verificar, quando A$ é "DAKOTA", LEFT$(A$,3) imprime os três caracteres mais à esquerda da cadeia, DAK. RIGHT$

(A$,3) produz os três caracteres mais à direita da cadeia, OTA. MID$(A$,2,3) fornece três caracteres da cadeia a partir do segundo, AKO.

ASC e CHR$

Em seguida observaremos duas outras palavras de tratamento de cadeias, ASC e CHR$. As letras, números e símbolos manipulados pelo nosso caracter possuem «códigos de caracteres». Estes são geralmente conhecidos por códigos ASCII; estas letras designam «American Standard for International Exchange», Código Normalizado Americano para Troca de Informação, constituindo o sistema de codificação mais usado para «alfanuméricos» (isto é, letras, algarismos e símbolos) ingleses.

O comando ASC é usado para obter o código ASCII de qualquer símbolo, e CHR$ é usado para transformar um código no símbolo correspondente. Os exemplos que se seguem devem esclarecer isto melhor:

PRINT ASC("A") produz 65
PRINT ASC ("B") produz 66
PRINT CHR$(65) produz A
PRINT CHR$(66) produz B

O leitor poderá ter notado que o Amstrad possui 31 caracteres que não podem ser impressos enquanto tais. Estes códigos, entre 1 e 31, são conhecidos por códigos de controlo e só podem ser apresentados carregando na tecla CTRL e numa das teclas de controlo (se tem curiosidade em saber o que estes códigos fazem, consulte a capa traseira do manual do Amstrad).

Estes códigos não podem ser normalmente impressos no visor. No entanto, se se iniciar uma linha de programas por PRINT CHR$(1) + + CHR$ (código do caracter) aparecerá no visor. O leitor já deve ter desenvolvido suficientemente as suas capacidades de programação para escrever um ciclo que imprime os 31 códigos referidos.

Comandos específicos

Vejamos agora um certo número de comandos específicos da forma como o Amstrad trata cadeias.

UPPER$, LOWER$

Estes comandos, raramente encontrados noutros dialectos BASIC, transformam uma cadeia para maiúsculas ou minúsculas. UPPER$ trans-

forma caracteres minúsculos em maiúsculos, enquanto LOWER$ realiza o contrário. Estes comandos podem ser úteis em programas que tratem muito texto e quando se pretende que as entradas do utilizador se encontrem num dado formato. Um outro uso útil destes comandos ocorre num programa complexo de tratamento de texto, quando as palavras especificadas pelo utilizador são sublinhadas e verificadas em termos de correcção da escrita.

SPACE$

O leitor não usará certamente esta ordem muitas vezes! SPACE$ cria uma cadeia de palavras com um dado comprimento; assim, SPACE$ (80) produz uma cadeia com 80 caracteres preenchida por espaços.

STRING$

Foi desta instrução que decorreu a SPACE$ já referida; aliás, STRING$ lança a dúvida sobre a necessidade de usar a SPACE$. A ordem STRING$ cria uma cadeia com um comprimento especificado mas contendo apenas um caracter definido. O formato da instrução é:

STRING$(N,"X")

onde N é o comprimento da cadeia e X o caracter que será usado na cadeia.

E se o leitor tenta descobrir coisas estranhas na BASIC «Locomotive» empregue no Amstrad, experimente colocar um caracter «&» à frente do número que especifica o comprimento da cadeia. Verificará que a cadeia é automaticamente prolongada em dois terços do comprimento especificado. Não sei muito bem para que possa servir esta descoberta...

INSTR

A INSTR procura uma cadeia no interior de outra. O formato deste comando muito útil é:

INSTR (N,"X","Y")

onde X é a cadeia que deve ser investigada, Y a cadeia onde aquela é procurada e N um número opcional que permite ao programador escolher onde deve começar a procurar.

Apresentamos alguns exemplos que esclarecerão melhor o leitor:

```
PRINT INSTR ("LOCOMOTIVE SOFTWARE", "SOFT")
12
PRINT INSTR (7, "LOCOMOTIVE SOFTWARE", "T")
7
```

CLEAR

O último comando que iremos tratar neste capítulo não se aplica apenas a cadeias, mas a todas as variáveis. CLEAR limita-se a apagar da memória do computador todas as variáveis e ficheiros.

Ordenação de cadeias

Para terminarmos este capítulo sobre tratamento de cadeias apresentamos um programa de ordenação de cadeias, que as colocará por ordem alfabética. A seguir ao programa apresentamos alguns exemplos de execução e, finalmente, uma breve discussão sobre divisão de cadeias.

```
10 ' ORDENAÇÃO
20 '
30 B = 0
40 INPUT ''TÍTULO'';T$
50 INPUT ''QUANTAS PALAVRAS'';T
60 DIM W$(T)
70 G = T
80 FOR A = 1 TO T
90 INPUT W$(A)
100 NEXT
110 CLS : PRINT T$
120 PRINT
130 Z = 1
140 B = Z + 1
150 IF B> G THEN 220
160 IF W$(B) > W$(Z) THEN 180
170 Z = Z + 1 : GOTO 140
180 Q$ = W$(Z)
190 W$(Z) = W$(B)
200 W$(B) = Q$
210 GOTO 170
220 PRINT W$(G)
230 G = G — 1
240 IF G > O THEN 130
```

PORSCHE
FORD
CHRYSLER
CITROEN
RENAULT
FIAT
FERRARI
BMW
DATSUN
VOLKSWAGEN
ROLLS ROYCE
LOTUS

FABRICANTES DE AUTOMÓVEIS

BMW
CHRYSLER
CITROEN
DATSUN
FERRARI
FIAT
FORD
LOTUS
PORSCHE
RENAULT
ROLLS ROYCE
VOLKSWAGEN

STABLEFORD
MAY
ADAMS
SILVERBERG
GREENE
HEMINGWAY

AUTORES FAVORITOS

ADAMS
GREENE
HEMINGWAY
MAY
SILVERBERG
STABLEFORD

Divisão de cadeias

O nome certo da divisão de cadeias é «concatenação». Esta operação permite criar novas cadeias a partir da original. Os comandos usados para a executar são os já citados LEFT$, RIGHT$ e MID$. Para somar cadeias, basta-nos colocar um sinal "+" entre elas (por exemplo, C$ = A$ + B$). A divisão de cadeias pode ser útil em programas que tratem textos — por exemplo, num programa de aventura.

9

ENTRADAS DE VÁRIOS TIPOS

Qualquer programa verdadeiramente interactivo permite que o utilizador introduza na máquina as informações sobre as quais o computador irá actuar. Estas entradas podem ser tão simples como o movimento de um punho (*joystick*) para controlo de um jogo de acção, ou tão detalhadas como a introdução do conteúdo de um relatório que deve ser tratado antes de enviado para uma impressora. Neste capítulo tentaremos tratar todos os comandos de entrada disponíveis no Amstrad.

INPUT

A instrução INPUT é seguida de uma cadeia ou de uma variável numérica; depois de ser introduzido o valor requerido, quer se trate de uma cadeia ou um número, deve-se carregar em ENTER. A INPUT não é usada em jogos de acção simplesmente porque demoraria muito tempo a introduzir deste modo a informação requerida. Observe a secção sobre o comando INKEY$ se quiser descobrir como se introduz informação sem usar a tecla ENTER.

Podemos colocar uma mensagem, entre aspas, entre as duas metades da instrução de INPUT — por exemplo, INPUT "INDIQUE DIMENSÃO DO CÍRCULO (1-20)";A. Isto significa que o programa pode fazer perguntas compreensíveis pelo utilizador, sem necessidade de recorrer a uma instrução PRINT extra.

LINE INPUT

Este comando é muito semelhante a INPUT, excepto que nos permite usar aspas na cadeia introduzida. LINE INPUT só é usada com variáveis de cadeia.

INKEY$

A instrução INKEY$ permite entradas rápidas, de um único caracter; isto faz-se geralmente tornando uma variável de cadeia igual à instrução INKEY$ — por exemplo, A$ = INKEY$.

Como a INKEY$ é muito rápida a ler o teclado, pode ser necessário acrescentar uma linha que impeça o programa de continuar até o utilizador carregar numa tecla. Use uma linha de programa do seguinte tipo:

```
10 A$ = INKEY$ : IF A$ = " " THEN 10
```

INKEY

Uma instrução muito semelhante à anterior, o comando INKEY produz os valores das teclas em que se carrega. A função INKEY é usada no formato seguinte:

```
IF INKEY(A) = N THEN . . .
```

O valor A é o número da tecla (note-se que não é o seu código ASCII, mas sim o número da tecla). O valor N é o estado da tecla; N pode ter um de cinco valores, cada um deles correspondente a uma diferente combinação de estados. Os cinco valores possíveis de N são explicados em seguida:

Valor de N	*Estado correspondente da tecla*
−1	A tecla especificada não está a ser premida.
0	A tecla está a ser premida, mas as teclas CTRL e SHIFT estão em repouso.
32	Estão a ser premidas simultaneamente a tecla especificada e a tecla SHIFT.
128	Estão a ser premidas simultaneamente a tecla especificada e a tecla CTRL.
160	Estão a ser premidas simultaneamente ambas as teclas SHIFT e CTRL e a tecla especificada.

Controlo por *joystick*

O Amstrad possui uma tomada para *joystick* na parte traseira, e toda a informação trazida através dela é controlada pelo comando BASIC JOY. Este comando é seguido por um zero e um um entre parêntesis, especificando o número do *joystick* usado. Se o leitor não possui a peça

especial que permite ligar até dois *joysticks* ao Amstrad, use sempre o número zero entre parêntesis.

O valor criado por JOY pode ser um de seis números diferentes — consulte a tabela seguinte:

Valor do «joystick»	*Acção*
1	Subir
2	Descer
4	Esquerda
8	Direita
16	Botão de tiro 2
32	Botão de tiro 1

Entradas «artísticas»

Depois de tratarmos os principais comandos de entrada de informação no Amstrad, observemos alguns programas que exemplificarão estas instruções. Os programas de demonstração baseiam-se num programa «artístico» que permite desenhar figuras no visor. Para as versões controladas por teclado (as duas primeiras), as teclas são: A, subir; Z, descer; O, esquerda; P, direita; C, alterar a cor de «caneta» (*pen*).

A primeira versão de *Artista* utiliza a instrução INPUT:

```
10 ' ARTISTA 1
20 ' USO DE INPUT
30 '
40 CLS
50 X = 40 : Y = 12 : P  = 1
60 LOCATE 1,25
70 PRINT " "
80 LOCATE 1,25
90 PEN 1
100 PRINT "COR:";P; : INPUT "INDIQUE DIRECÇÃO"; A$
110 IF A$ < > "A" AND A$ < > "Z" AND A$ < > "O" AND
    A$ < > "P" AND A$ < > "C" THEN 60
120 IF A$ = "A" AND Y > 1 THEN Y = Y — 1
130 IF A$ = "Z" AND Y < 24 THEN Y = Y + 1
140 IF A$ = "O" AND X > 1 THEN X = X — 1
150 IF A$ = "P" AND X < 40 THEN X = X + 1
160 IF A$ = "C" THEN P = P + 1 : IF P = 4 THEN P = 0
170 PEN P
180 LOCATE X,Y
190 PRINT CHR$(143)
200 GOTO 60
```

A segunda versão de *Artista* utiliza o comando INKEY$:

```
10 ' ARTISTA 2
20 ' USO DE INKEY$
30 '
40 CLS
50 X = 20 : Y = 12 : P = 1
60 LOCATE 1,25
70 PEN 1
80 PRINT "COR: ";P
90 A$ = INKEY$
100 IF A$ < > "A" AND A$ < > "Z" AND A$ < > "O" AND
    A$ < > "P" AND A$ < > "C" THEN 90
110 IF A$ = "A" AND Y > 1 THEN Y = Y — 1
120 IF A$ = "Z" AND Y < 24 THEN Y = Y + 1
130 IF A$ = "O" AND X > 1 THEN X = X + 1
140 IF A$ = "P" AND X < 40 THEN X = X + 1
150 IF A$ = "C" THEN P = P + 1 : IF P = 4 THEN P = 0
160 PEN P
170 LOCATE X,Y
180 PRINT CHR$ (143)
190 GOTO 60
```

A versão final de *Artista* permite aos utilizadores introduzirem informação através do *joystick*:

```
10 ' ARTISTA 3
20 ' USO DO JOYSTICK
30 '
40 CLS
50 X = 20 : Y = 12 : P = 1
60 LOCATE 1,25
70 PEN 1
80 PRINT "COR: ";P
90 A = JOY(0)
100 IF A = 1 AND Y > 1 THEN Y = Y — 1
110 IF A = 2 AND Y < 24 THEN Y = Y + 1
120 IF A = 4 AND X > 1 THEN X = X — 1
130 IF A = 8 AND X < 40 THEN X = X + 1
140 IF A = 16 THEN P = P + 1 : IF P = 4 THEN P = 0
150 PEN P
160 LOCATE X,Y
170 PRINT CHR$(143)
180 GOTO 60
```

Outros comandos de teclas

Parece ser agora o melhor momento de tratarmos detalhadamente os comandos que afectam as próprias teclas.

SPEED KEY

O comando SPEED KEY define a velocidade de repetição de cada tecla. É usado com dois números, correspondendo o primeiro ao atraso antes do início da repetição (medido em unidades de 0,02 segundos) e o segundo à velocidade de repetição (medida nas mesmas unidades). Tenha cuidado ao usar esta instrução, porque pode dificultar a escrita no Amstrad. Os valores automáticos da velocidade das teclas são 10,10.

KEY DEF

A instrução KEY permite-nos definir uma tarefa especial para uma tecla. Vejamos um exemplo do uso de KEY DEF:

KEY DEF 47,1,65

Esta instrução leva a tecla número 47 (Z) a imprimir a maiúscula A. O primeiro número a seguir à palavra chave indica o número da tecla que está a ser redefinida. O segundo número é um um ou um zero, e indica se se pretende ou não que a tecla funcione com repetição (1 mantém a repetição, 0 anula-a). O último número indica a nova definição da tecla; no nosso exemplo a nova definição é 65, o código ASCII do caracter A.

KEY

Este comando permite-nos dar uma função completamente nova a uma tecla — não se altera o caracter que a tecla imprime, mas altera-se a sua função global.

Por exemplo, KEY 140, "CLS" + CHR$(13) define a tecla ENTER de tal modo que quando se carrega juntamente em CTRL e ENTER aparece a palavra CLS (o CHR$(13) garante que o comando é introduzido no computador e é executado). Escreva a linha de programa acima indicada e experimente-a até compreender o seu funcionamento.

O programa de demonstração que se segue altera duas teclas, a tecla zero e a tecla um do teclado numérico, de modo a executarem vários comandos cada. Execute o programa, que imprime alguns asteriscos no visor, e carregue em seguida na tecla zero — o visor é limpo e o

programa é listado. Em seguida carregue na tecla um, o que leva o programa a ser executado e a ouvir-se um som.

```
10 '   DEMONSTRAÇÃO DE TECLAS
20 '
30 KEY 128, ''CLS'' + CHR$(13) + ''LIST'' + CHR$(13)
40 KEY 129, ''SOUND 1,180,30,15'' + CHR$(13) + ''FOR T = 1
   TO 1000 : NEXT'' + CHR$(13) + ''RUN'' + CHR$(13)
50 CLS
60 PRINT ''TECLAS''
70 FOR T = 1 TO 20
80 PRINT '' * * * * * * * * * * * * * * * * * * * * ''
90 NEXT
```

Convém notar algumas limitações do uso da instrução KEY. Em primeiro lugar só se pode usar o grupo de 32 caracteres de expansão com códigos entre 128 e 159. Em segundo lugar, as cadeias que contêm as instruções definidas e os códigos de caracteres — como CHR$(13) — não podem ter um comprimento superior a 120 caracteres.

10

COMANDOS DE AUXÍLIO À PROGRAMAÇÃO

Vamos apresentar agora um certo número de instruções que tornarão a programação um pouco mais fácil.

RENUM

Observemos primeiro a instrução que provavelmente nos será mais útil — RENUM. Este comando permite-nos renumerar as linhas de programa começando no número que desejarmos e com o passo mais adequado. A sintaxe desta instrução é:

RENUM (novo número de linha), (antigo número de linha), (incremento)

Isto pode parecer bastante complexo, mas na prática não o é. Se o seu programa tiver quatro linhas começando na 10 e o leitor escrever RENUM 100,20,10, as linhas originais passarão a 10, 100, 110 e 120; o processo de renumeração começa na segunda linha do programa, deixando a linha 10 como se encontra, e renumera as três linhas restantes em passos de 10 a partir da linha 100.

Pode-se igualmente usar o comando para criar mais espaço num bloco de linhas onde já não é possível incluir uma nova linha. Por muito bem concebido que um programa seja, haverá sempre necessidade de acrescentar algumas linhas. Por outro lado, quando se termina a listagem, pode-se dar uma melhor aparência ao programa renumerando-o em passos de 10, começando na linha 10. Para tal basta escrever RENUM e carregar em ENTER; escrever RENUM equivale a RENUM 10,1,10.

DELETE

Esta instrução permite-nos eliminar blocos de linhas; por exemplo,

DELETE 100-400 elimina todas as linhas de programa cujos números se encontram entre aqueles, além das próprias linhas 100 e 400.

Tal como a ordem LIST, DELETE pode ser usada com um só número. Por exemplo, DELETE 1000 - eliminará todas as linhas a partir da 1000, inclusive, até ao final do programa; DELETE −350 eliminará todas as linhas desde o início até à linha 350, inclusive.

FRE

A instrução FRE tem várias utilidades, mas a mais importante como auxiliar de programação é quando usada para indicar ao utilizador a quantidade de memória que ainda se encontra livre. Quando se liga a máquina, podemos escrever PRINT FRE(0) e obteremos a resposta de 43533; este valor é dado em "bytes" — noutros casos usa-se muito o termo "K", indicando "Kbyte", ou 1024 bytes. Esta resposta indica que dispomos mais de 43K livres para programação, o que é bastante.

Assim, se em qualquer momento em que o leitor esteja a escrever um programa vasto tiver curiosidade em saber o que lhe resta de memória, pode descobrir a resposta escrevendo simplesmente PRINT FRE(0). Se possuir um Amstrad 6128 terá muito mais memória disponível, se bem que esta só possa ser usada indirectamente. Leia o capítulo sobre o 6128 para obter mais detalhes.

AUTO

O comando AUTO evita o trabalho de escrevermos números de linha. Se quisermos usar números de linha regulares, AUTO pode permitir-nos poupar algum tempo. AUTO é seguida de dois números, o correspondente à linha inicial e a dimensão do passo. Por exemplo, AUTO 345,10 produzirá números a partir de 345 em passos de dez.

TRON e TROFF

Estas duas instruções são comandos de *trace*, ou seja, utilizadas para seguir o desenrolar de um programa. É assim possível conhecer o número da linha que vai ser executada em qualquer momento. O programador pode assim saber se o programa executa um salto no momento adequado e para o local certo. O *trace* é ligado com TRON e TROFF.

EDIT e montagem de linhas

O leitor certamente cometerá erros ao escrever programas ou ao copiá-los de um livro. Muitas linhas são bastante curtas, e não é difícil

escrevê-las de novo — mas o que acontece quando se trata de uma linha monstruosa que demora bastante tempo a escrever? Felizmente o Amstrad possui uma instrução EDIT. Escreva a linha que se segue e execute-a:

```
10 FOR T = 1 TO 100 : PRINT "DEMONSTRAÇÃO : PRINT :
   NEXT
```

Obterá uma mensagem de erro "NEXT em falta na linha 10". O que de facto aconteceu é que faltam as aspas finais, e o computador «julga» que tudo o que se encontra depois da instrução PRINT é um texto para escrever; não avalia portanto a instrução NEXT.

Para remediar isto, escreva EDIT 10 para trazer a linha para a área do visor onde pode alterá-la. Guie o cursor ao longo da linha usando as teclas de cursor (as que se encontram acima das teclas numéricas) até atingir o último O de DEMONSTRAÇÃO. Acrescente as aspas em falta — e a linha ficará corrigida. Carregando na tecla ENTER colocamos a linha corrigida na memória do computador. Se desejamos libertar-nos de um caracter em excesso, podemos colocar o cursor um caracter para a direita e carregar na tecla DEL, ou colocar o cursor sobre o caracter errado e carregar em CLR.

Se em qualquer momento nos enganarmos ao corrigir uma linha, podemos carregar na tecla ESC e voltaremos a obtê-la na sua forma original, podendo começar a alterá-la de novo.

Existe uma outra forma de montar uma linha, a qual é explicada no primeiro capítulo do manual Amstrad.

11

O AMSTRAD E A SUA UTILIDADE

O seu Amstrad CPC 464, com o seu gravador de *cassettes* incorporado, não se encontra felizmente sujeito aos problemas que abundam noutros computadores domésticos. Neste capítulo discutiremos o modo de carregar e gravar programas em *cassette*, tratando os vários comandos que actuam sobre ficheiros em *cassette*. Se o leitor possui um 664 ou um 6128, poderá ler este capítulo e passar depois ao Capítulo 12.

SPEED WRITE

Gravar e carregar um programa não é uma tarefa complexa no CPC 464. O utilizador pode escolher entre duas velocidades de transferência diferentes; a que escolher para gravar em fita deve ser igualmente usada para carregar o mesmo programa. Selecciona-se a velocidade usando o comando SPEED WRITE — se esta instrução for seguida de um zero, a transferência é feita a 1000 baud; se se usa um, a transferência é realizada a 2000 baud. Quando o CPC 464 é ligado passa automaticamente para uma velocidade de 1000 baud.

SAVE e LOAD

Coloque a *cassette* no gravador (é melhor usar *cassettes* com seis ou dez minutos por lado), verificando se foi rebobinada para o lado certo. Escreva SAVE «Nome» (o nome pode ter catorze letras de comprimento) e carregue na tecla ENTER. O Amstrad indica-nos então que devemos carregar nas teclas de gravação do gravador, e depois em qualquer tecla do computador. A gravação inicia-se imediatamente.

Carregar um programa é uma operação semelhante. Escreva LOAD seguido do nome do programa entre aspas ou, se apenas quiser carregar o primeiro programa existente na *cassette*, escreva LOAD «». O Amstrad imprime uma mensagem de instruções semelhante à anterior.

O CPC 464 procura então na fita o primeiro programa correctamente gravado (ou o ficheiro cujo nome foi indicado) e carrega-o em seguida.

Uma alternativa ao uso da instrução LOAD é RUN «». Esta instrução leva o programa a entrar automaticamente em execução depois de carregado.

CAT

Para verificar o conteúdo de uma *cassette*, pode usar o comando CAT. Este CATaloga todo o conteúdo da *cassette*, indicando se existe um ficheiro de programa BASIC (indicado por um cifrão), um ficheiro de texto ASCII (indicado por um asterisco), um ficheiro binário (indicado por &) ou um programa BASIC protegido (indicado pelo sinal de percentagem).

MERGE

Podem-se misturar dois programas de modo a formar um programa único; isto é particularmente útil quando, digamos, se desenvolveu uma rotina útil guardada em fita que se pretende incorporar em diversos programas. O formato a usar é simplesmente MERGE «nomes». Os efeitos do comando MERGE são, no entanto, um pouco mais complexos. Em primeiro lugar, os ficheiros protegidos não podem ser «misturados», como sugere o nome do comando — são eliminadas todas as variáveis, ficheiros e funções; e finalmente é realizado um RESTORE, passando o indicador dos dados para o início das declarações DATA.

Se deseja que o programa ainda em memória não seja afectado pela mistura, terá de o renumerar de modo a não possuir os mesmos números de linha do programa a misturar.

CHAIN

Este comando é semelhante a MERGE, exceptuando o facto de substituir o programa em memória em vez de se misturar com ele. Depois de CHAIN utiliza-se um nome e um número opcional; este número indica o número da linha por onde o programa deverá entrar em execução se assim se quiser.

CHAIN MERGE combina os dois comandos referidos, de tal modo que um programa em fita é misturado com o que já se encontra na memória, entrando automaticamente em execução pela linha especificada na instrução CHAIN MERGE. Se não for mencionado qualquer número de linha em nenhum destes comandos, o programa começa pela menor linha existente.

Antes de abandonarmos o tema da gravação e carga de programas, vale a pena notar que, no caso de ser colocado o caracter «!» à frente do nome do ficheiro, são eliminadas as mensagens produzidas pelo CPC 464.

Ficheiros de dados

O resto dos comandos de carga e gravação tem a ver com ficheiros de dados. Para construir um ficheiro de dados, é necessário abrir um ficheiro com um nome, enviar os dados para a *cassette*, e fechar em seguida o ficheiro. Para carregar de novo o ficheiro para o CPC 464, deve-se abrir um ficheiro com o nome do pretendido, introduzir os dados a partir da fita, verificar se foi atingido o final do ficheiro e fechá-lo.

Isto pode parecer complicado, mas é mais simples do que o leitor pensa, particularmente quando se tem em conta os nomes dos comandos usados. É possível usar várias instruções, mas por uma questão de facilidade apenas trataremos de uma forma de construir ficheiros de dados.

Gravar um ficheiro

OPENOUT "Nome"
PRINT # 9, todos os dados a ser passados para a *cassette*
CLOSEOUT

Carregar um ficheiro

OPENIN "Nome"
INPUT # 9, todos os dados a carregar da *cassette*
IF EOF-1 THEN CLOSEIN (este último comando verifica a presença da marca de fim de ficheiro. Se esta é encontrada, o ficheiro é fechado)
CLOSEIN

Base de dados

Para demonstrarmos o uso dos comandos para ficheiros de dados, apresentamos um programa que grava e carrega ficheiros deste tipo. Foi mantido deliberadamente simples a fim de mostrar mais claramente a acção dos comandos em causa. Introduza até 50 dados, e depois grave o ficheiro em *cassette* com um nome. Este ficheiro poderá ser recuperado mais tarde, bastando usar o modo *load*.

Não seria difícil melhorar este programa — de facto, foi escrito de tal modo que facilite essa tarefa. Por que não acrescentar a possibilidade de procurar uma cadeia ou aumentar o número de ficheiros tratados pelo computador? Uma tarefa simples consistiria em passar os dados para a impressora.

Um último aspecto, relativo à apresentação do visor. Quando é impresso o primeiro elemento do ficheiro, é necessário carregar numa tecla para manter a impressão. Este parece ser o melhor método de apresentar até 50 dados, permitindo o exame pelo utilizador.

```
10 ' DEMONSTRAÇÃO BASE DE DADOS
20 '
30 DIM A$(50)
40 PRINT : PRINT
50 PRINT "1 INTRODUZIR DADOS, 2 GRAVAR DADOS,
   3 CARREGAR DADOS, 4 IMPRIMIR NO VISOR"
60 INPUT G: IF G > 4 OR G < 1 THEN PRINT : GOTO 50
70 ON G GOSUB 120,250,370,490
80 GOTO 40
90 '
100 ' INTRODUÇÃO DE DADOS
110 '
120 CLS
130 INPUT "QUANTAS ENTRADAS";N
140 IF N > 50 OR N < 1 THEN PRINT : GOTO 130
150 PRINT : PRINT
160 FOR T = 1 TO N
170 PRINT "INDIQUE DADO N.";T
180 INPUT "        ";A$(T)
190 PRINT : PRINT
200 NEXT
210 RETURN
220 '
230 ' GRAVAR DADOS
240 '
250 CLS
260 INPUT "NOME DO FICHEIRO (ATÉ 14 CARACTERES)";F$
270 PRINT : PRINT
280 OPENOUT F$
290 FOR T = 1 TO 50
300 PRINT # 9,A$(T)
310 NEXT
320 CLOSEOUT
330 RETURN
```

```
340 '
350 ' CARREGAR DADOS
360 '
370 CLS
380 INPUT ''FICHEIRO A CARREGAR'';F$
390 PRINT : PRINT
400 OPENIN F$
410 FOR T = 1 TO 50
420 INPUT # 9,A$(T)
430 NEXT
440 CLOSEIN
450 RETURN
460 '
470 ' IMPRIMIR DADOS
480 '
490 CLS
500 FOR T = 1 TO 50
510 PRINT A$(T)
520 WHILE INKEY$ = '' '' : WEND
530 NEXT
540 RETURN
```

Sugestões

Se bem que o uso das *cassettes* com o CPC 464 seja bastante simples, várias coisas podem danificar o conteúdo das *cassettes*. Convém portanto que o leitor siga as sugestões aqui dadas e mantenha as suas *cassettes* em bom estado.

1) NUNCA guarde as suas *cassettes* perto de um aparelho que produza um forte campo electromagnético, como, por exemplo, a televisão. Isto pode eliminar completamente o que se encontra gravado nas *cassettes*, ou pelo menos corromper os ficheiros.

2) NUNCA guarde as suas *cassettes* perto de uma área anormalmente quente ou fria, como por exemplo um radiador ou uma janela aberta. As temperaturas extremas podem afectar bastante as *cassettes*.

3) Etiquete SEMPRE as suas fitas, indicando o nome do programa gravado e, se possível, utilize o contador existente no gravador para identificar com exactidão o ponto da fita onde o programa se encontra.

4) Use SEMPRE a instrução CAT para conhecer o que se encontra em cada *cassette*... é muito melhor do que tentar adivinhar!

12

USO DE DISCOS

Se o leitor possui um Amstrad CPC 664 ou 6128, não necessita obviamente de ligar um *drive* ao computador. Basta não ler os três parágrafos que se seguem, e passar directamente à secção intitulada «Os discos».

Antes de tentar ligar o *drive* de disco ao seu 464, verifique bem se as alimentações tanto deste como do *drive* se encontram desligadas. Aproximadamente a meio do computador, na sua parte traseira, encontra-se um ligador identificado pelas palavras FLOPPY DISC. Introduza o *interface* firmemente neste ligador. A outra extremidade da fita achatada deve ser ligada ao próprio *drive* (a ficha a cerca de dez centímetros do final da fita serve para ligar um segundo *drive*).

Depois de ter ligado o *drive*, ligue a unidade de disco e depois o computador. O disco deve ser sempre ligado antes do computador. Por outro lado, deve-se verificar se o disco foi retirado do *drive* antes de este ser ligado.

É melhor colocar a unidade de disco à direita do monitor, em vez da sua esquerda, dado que o lado esquerdo do monitor parece produzir uma interferência eléctrica que leva o controlador de disco a funcionar de modo menos fiável. É importante também que o *drive* e os discos sejam mantidos longe de campos magnéticos, como os produzidos por altifalantes de alta fidelidade.

Os discos

Verificará que os discos de três polegadas usados pelo seu computador possuem uma patilha no lado esquerdo de cada lado. Esta patilha é usada para impedir que se destrua a informação guardada em disco.

A figura 1 mostra a aparência do disco.

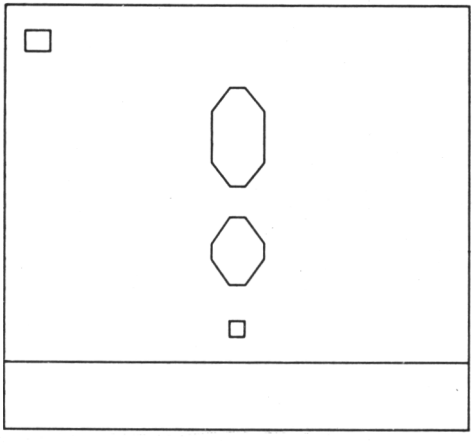


Fig. 1

Quando o *drive* é ligado, acende-se uma luz verde: é o indicador de funcionamento do *drive*. Um pouco mais acima, no lado direito do painel dianteiro do *drive*, encontra-se um botão que é usado para ejectar os discos. Do outro lado existe uma luz vermelha. Este indicador realiza duas tarefas. Acende-se quando está apenas ligado um *drive*, quando está a ser gravada informação e quando esta está a ser lida. Se possuímos dois *drives*, a luz vermelha do segundo *drive* (conhecido por *drive* B, sendo o primeiro designado convencionalmente por *drive* A) está continuamente acesa, de tal modo que o utilizador nunca esquece qual é o *drive* B.

AMSDOS

AMSDOS significa «Sistema Operativo de Discos Amstrad». Garante que os ficheiros BASIC podem aceder o disco exactamente da mesma forma (se bem que muito mais depressa, evidentemente) que acediam o gravador de *cassettes*.

Todos os comandos BASIC (exceptuando CAT, que possui um significado especial, que descobriremos daqui a pouco) funcionam com o *drive* de disco e a *cassette*, tendo exactamente o mesmo efeito.

Isto significa que podemos usar todas as instruções seguintes com os mesmos resultados tanto no *drive* como no gravador de *cassettes*:

LOAD, RUN, SAVE, CHAIN, MERGE, CHAIN MERGE, OPENIN, OPENOUT, CLOSEIN, CLOSEOUT, EOF, INPUT # 9, LINE INPUT # 9, WRITE # 9, LIST # 9

Para apagar um ficheiro do disco, faça o seguinte. Considere que o nome do ficheiro é *turtle*.

Escreva directamente o seguinte:

F$="turtle":|era,@f$ < ENTER >

Se precedermos isto com CAT, e escrevermos também CAT à frente, verificaremos que *turtle* desapareceu do «catálogo».

Para alterar o nome de um ficheiro em disco, execute os passos seguintes. Considere que o nome original do ficheiro é *turtle* e quer alterá-lo para *terrapin*.

Escreva directamente o seguinte:

one$="turtle":two$="terrapin":|ren,@two$,@one$ < ENTER >

CAT mostrará novamente que *turtle* desapareceu do catálogo, passando a *terrapin*.

Já se mencionou que CAT se comporta de forma diferente conforme é usada para discos ou *cassettes*. A CAT usada com o gravador limita-se a ler a fita, lendo cada nome ao contrário. No caso dos discos, CAT ordena alfabeticamente todos os ficheiros em disco e imprime-os, juntamente com uma indicação do comprimento do ficheiro e uma mensagem indicando o espaço livre em disco.

A mensagem BAD COMMAND indica-nos que escrevemos um nome demasiado comprido ou que contém espaços ou pontuação incorrectos. TYPE MISMATCH indica que esqueceu as aspas e SYNTAX ERROR significa que escreveu RUN incorrectamente.
Se obtiver a mensagem

drive A : read fail
Retry, Ignore or Cancel?

significa que, por alguma razão (disco errado, não formatado ou formatado incorrectamente, ou dados corruptos), o computador foi incapaz de ler do disco actualmente no *drive*.

Se está habituado a *cassettes*, talvez fique surpreendido por ver a familiar mensagem «Press PLAY then any key» aparecer no visor quando indica ao computador que deve carregar um programa. Isto deve-se ao facto de o computador ignorar o *drive* de discos, por este não ter sido ligado ou o leitor ter usado o comando |TAPE (*barra tape*) para indicar que deseja escrever e ler do gravador em vez do *drive*. Para inverter este efeito basta escrever |DISC e experimentar de novo (note que apesar de ser possível escrever SPEED WRITE enquanto tem um disco ligado, não se trata de um comando válido para disco, apenas funcionando com o gravador; o sistema de disco já escreve tão depressa neste quanto pode).

O tão usado sistema operativo CP/M (que significa «Control Program Microcomputer») é fornecido juntamente com o sistema. O nome indica claramente para que serve.

O CP/M foi inventado por Gary Kildall, há cerca de dez anos, quando pretendia ligar uma unidade de disco ao seu computador pessoal, que era certamente muito primitivo quando comparado aos que hoje existem. Não havia qualquer maneira de realizar a ligação, pelo que Gary resolveu desenvolver um sistema próprio. Pensou depois que talvez algum outro possuidor de microcomputadores pudesse estar interessado no sistema que desenvolvera.

Gary trabalhava para um fabricante de semicondutores, a Intel, e escrevera de facto parte do seu sistema como trabalho executado para esta empresa. Perguntou à Intel se pretendia comercializá-lo. Foi-lhe respondido que os microcomputadores não eram importantes e que a Intel não tinha qualquer interesse em envolver-se em produtos de brinquedo. «Pode fazer o que quiser com esse sistema», foi-lhe dito.

Kildall fez. Começando por praticar vendas pelo correio, vendeu de facto o sistema a outros utilizadores de microcomputadores. Mas o que não previra foi o enorme interesse que o seu sistema despertou. O mercado que descobriu foi tão grande que abandonou o seu trabalho e fundou uma companhia própria — a Digital Research — para vender o produto. Actualmente é um multimilionário e o *software* produzido pela Digital Research continua a ter um enorme êxito no mercado.

O êxito de Kildall baseou-se em duas coisas. Em primeiro lugar, o seu sistema de controlo funcionava de facto, dando aos possuidores de microcomputadores um modo de ligar os discos às suas máquinas. Além disto, e inadvertidamente, conseguira criar um sistema *standard* de controlo de discos. Os discos produzidos num computador que utilize CP/M funcionam com qualquer outro computador que utilize também esse sistema. Existem actualmente mais de 3000 produtos em CP/M, e apesar de necessitarem de algum trabalho para poderem ser usados no Amstrad 464 ou 664, devido a problemas de memória (mas não no 6128), porque o sistema exige 64 K em vez de cerca de 39 K disponíveis no 464 e no 664, vários destes produtos estão hoje disponíveis no Amstrad.

O utilizador sabe que está a trabalhar em CP/M porque vê impresso o seguinte:

CP/M 2.2 — Amstrad Consumer Electronics plc.

No 6128 usa-se a versão 3.0 em vez de 2.2. Segue-se, na linha seguinte, A > , o que indica que o sistema CP/M está pronto a ser utilizado. É o equivalente CP/M à mensagem READY da BASIC.

Depois de estarmos em CP/M, o seu Amstrad deixará de compreender a linguagem BASIC, e qualquer tentativa de introduzir uma instrução nesta linguagem provocará simplesmente a escrita da palavra no visor, seguida de um ponto de interrogação, que indica que o sistema não a compreende.

No entanto, e apesar de não compreender a BASIC, o CP/M possui um vocabulário próprio. Neste incluiu-se DIR (directório), que obtém uma lista do material existente no disco, e FORMAT, que prepara um disco para poder ser usado (volta-se ao sistema do computador escrevendo AMSDOS e carregando na tecla ENTER).

Cópia de programas de um disco para outro

Um só «drive»

O leitor chamou o CP/M escrevendo |CPM < ENTER >. Vê agora no visor o pedido de entrada, A >. Escreva DISCCOPY; o *drive* produz um zumbido algumas vezes, antes de aparecer a mensagem seguinte:

Please insert source disc into drive A then press any key

(por favor coloque o disco fonte no *drive* A e depois carregue numa tecla). Remova o disco CP/M e substitua-o por aquele que pretende copiar. Depois responda às perguntas que o sistema lhe fez. Parte do disco será armazenado em RAM; depois ser-lhe-á pedido que introduza o disco «destino», para que essa parte do disco original possa ser gravada nele. Ser-lhe-á então pedido que introduza novamente o disco fonte, para que a secção seguinte possa ser copiada em RAM.

Dois «drives»

Usando dois *drives*, o processo é mais simples. Em vez de DISCCOPPY, usa-se o comando COPYDISC. Basta colocar o disco fonte num dos *drives*, e o disco do destino no outro *drive*. O computador imprimirá as mensagens que o auxiliarão nesta tarefa.

Verificar os discos copiados

Se quiser verificar a cópia feita, pode usar CHKDISC (se possuir dois *drives*). As mensagens impressas indicar-lhe-ão o que deve fazer.

UTILITÁRIOS CP/M

As possibilidades de cópia e verificação dos discos são apenas algumas das facilidades permitidas pelo seu sistema CP/M. Vejamos um resumo das outras:

AMSDOS.COM	reenvia ao AMSDOS e à BASIC (oposto de ICP/M)
ASM.COM	assemblador 8080
BOOTGEN.COM	copia os sectores de inicialização e configuração de um disco para outro
CLOAD.COM	copia um ficheiro em *cassette* para disco
CSAVE.COM	copia um ficheiro de disco para *cassette*
CHKDISC.COM	compara dois discos (necessita de dois *drives*)
COPYDISC.COM	copia um disco para outro (necessita de dois *drives*)
DDT.COM	Dynamic Debugging Tool, para detecção de erros em 8080
DISCCHK.COM	compara dois discos (usando um único *drive*).
DISCCOPY.COM	copia um disco para outro (usando um único *drive*)
DUMP.COM	imprime um ficheiro no visor, em hexadecimal
ED.COM	editor de texto
FILECOPY.COM	copia ficheiros de um disco para outro (usando um único *drive*)
FORMAT.COM	formata disco
LOAD.COM	lê um ficheiro no formato HEX Intel; produz um ficheiro .COM
MOVECPM.COM	constrói um sistema CP/M de uma dada dimensão (usado para reservar espaço para RSX's)
PIP.COM	Peripheral Interchange Program; copia ficheiros em disco entre outros periféricos
SETUP.COM	altera parâmetros no sector de configuração.
STAT.COM	indicação de estado dos ficheiros, discos, utilizadores e IOBYTE (pode também ser usado para alterar IOBYTE)
SUBMIT.COM	aceita comandos CP/M de ficheiro (em vez de os aceitar do teclado)
SYSGEN.COM	escreve o sistema CP/M nas pistas de sistema
XSUB.COM	usado com SUBMIT.COM para entrada de programa em *buffer*.

```
#FFFF ┌ - - - - - - - - - ┐      #C000 . . . #FFFF
      ¦ BIOS ROM          ¦
#C000 ├ - - - - - - - - - ┤
      ¦ BIOS STACK        ¦
#BECO ├ - - - - - - - - - ┤
      ¦ BIOS EXTENDED     ¦
      ¦  JUMPBLOCK        ¦
#BE80 ├ - - - - - - - - - ┤
      ¦ FIRMWARE & BIOS   ¦
      ¦  VARIABLES        ¦      RSX/s
#AD33 ├ - - - - - - - - - ┤
      ¦ BIOS JUMPBLOCK    ¦
#AD00 ├ - - - - - - - - - ┤
      ¦ BDOS              ¦
#9F06 ├ - - - - - - - - - ┤
      ¦ CCP               ¦
#9700 ├ - - - - - - - - - ┤
      ¦ TPA               ¦      #0000 . . . #3FFF
#0100 └ - - - - - - - - - ┘
```

Fig. 2 — Mapa de memória CP/M (tão alto quanto possível)

Logo

Além do CP/M, o seu disco contém a versão Logo da Digital Research. Para invocar a Logo em BASIC, rode o disco de sistema (usando a sua cópia de segurança) para o lado dois (ficando, portanto, visíveis as palavras DR LOGO) e escreva o comando |CPM. O *drive* incia o seu zumbido habitual, aparecendo no visor uma mensagem indicando que a Logo foi carregada.

A Logo foi desenvolvida por uma equipa de matemáticos sob a direcção de Seymour Papert (autor do mais inportante texto Logo, *Mindstorms*) a fim de facilitar a introdução das crianças nas tarefas de programação; mas também os adultos apreciam esta linguagem, em particular devido à elegância de muitos dos padrões desenhados pela tartaruga ao mover-se no visor.

Primitivas Logo

Os blocos constitutivos da Logo são conhecidos por *primitivas*, e o leitor pode fazer muitas coisas com apenas alguns deles.

Experimente o seguinte, depois de ter carregado a Logo (não se esqueça de carregar em ENTER depois de cada uma destas instruções):

```
forward 100
back 100
right 90
forward 100
right 90
forward 130
left 80
back 100
```

Quatro das palavras mais importantes são as que movem a tartaruga pelo visor (*forward* e *back*) e as que controlam as suas curvas (*right* e *left*). As primitivas de movimento devem ser seguidas de um número que diz à tartaruga quantos «passos» deve dar. As que controlam curvas são seguidas de um número que corresponde à rotação em graus que a tartaruga deve executar em relação à direcção em que se encontra.

Pode-se usar a primitiva *cs* para limpar o visor. Depois de ter feito isto, escreva a linha seguinte, terminando-a com ENTER:

```
repeat 4 [fd 110 rt 90]
```

Descobrirá que isto desenha um quadro no visor. A primitiva *repeat* leva a tartaruga a repetir os comandos dentro de parêntesis o número de vezes indicado antes destes.

No seu manual encontrará uma lista completa das primitivas Logo. No entanto, para que possa compreender a enorme flexibilidade da Logo, vejamos cinco interessantes desenhos que pode criar no seu visor, usando apenas os comandos que indicámos até agora:

```
1) REPEAT 45 [FD 130 RT 92]
2) REPEAT 20 [FD 120 RT 182 FD 240 RT 182 FD 120]
3) REPEAT 30 [FD 120 RT 190 FD 240 RT 190 FD 120]
4) REPEAT 100 [FD 100 RT 93 FD 10 RT 97 FD 101 RT 180]
5) REPEAT 70 [FD 130 RT 181 FD 260 RT 182 FD 128]
```

Se quiser aumentar os seus conhecimentos sobre a Logo, poderá consultar outros textos sobre o assunto. Como os diversos dialectos da Logo são bastante semelhantes, verificará que quase todos os programas que encontrar poderão ser usados no seu sistema sem grandes modificações.

13

JOGOS! JOGOS! JOGOS!

Vejamos agora uma interessante selecção de jogos que certamente darão prazer ao leitor. As listagens apresentadas cobrem todo o espectro de jogos de computador, incluindo os de aventuras, acção, estratégia e lógica.

A escrita de jogos é um modo excelente de melhorar as suas capacidades de programação. O simples acto de cópia destes programas para o seu computador ensinar-lhe-á alguma coisa sobre a concepção de programas e algumas sugestões sobre como poderá apresentar os seus próprios jogos. A introdução a cada jogo dá algumas indicações sobre a forma como se joga — e ganha!

Madrugada vermelha

Enquanto se move sobre as suaves ondas do mar Marciano, tudo parece calmo. Você é o comandante de uma das naves de patrulha da Federação e deve investigar a presença na área de um bando de mutantes que ameaçam atacar a qualquer momento. A sua nave, humilde relativamente às outras da Federação, dispõe apenas de 50 torpedos de plasma... convirá portanto usá-los com cuidado! De qualquer modo, parece não haver necessidade deles imediatamente... Mas... que é aquilo no horizonte? Sim, é o inimigo.

Você agarra o punho de controlo; dentro em pouco estará a travar uma dura batalha. Boa sorte.

Não poderíamos ter uma colecção de jogos que não incluísse pelo menos um de acção, não é? Este é um tanto difícil de jogar devido à forma como os mutantes se deslocam. Uma pontuação de mais de mil justificará já o seu orgulho!

```
10 ' MADRUGADA VERMELHA
20 '
30 CLS
40 BORDER 6
50 X=12
60 PEN 3
70 FOR T = 1 TO 40: LOCATE T,25 : PRINT CHR$(244);:NEXT
80 LOCATE 1,1
90 PRINT " ///// MADRUGADA VERMELHA ///// "
100 C=1 : D=1
110 B=2 : A=INT(RND*30)+5
120 PEN 1
130 ' CICLO PRINCIPAL
140 LOCATE A,B : PRINT " ";
150 A = A+C : B=B+D
160 LOCATE A,B : PRINT CHR$(225);
170 IF A > 39 THEN C= -1
180 IF A< 2 THEN C=1
190 IF B< 3 THEN D=1
200 IF B > 21 THEN D=-1
210 Z=JOY(0) : IF Z=16 THEN MF=MF+1 : K=22
220 M=X
230 LOCATE X,23 : PRINT " ";
240 IF Z=4 AND X > 1 THEN X=X-1
250 IF Z=8 AND X < 40 THEN X=X+1
260 LOCATE X,23 : PRINT CHR$ (229);
270 IF K< 2 THEN 320
280 LOCATE M,K : PEN 2: PRINT ":";: PEN 1 : K = K-1
290 IF K=B AND X=A THEN SOUND 1,80,10 : SC=SC+137 :
    K=0 : LOCATE A,B : PRINT "X";: FOR T=-1 TO 20:
    NEXT :LOCATE A,B : PRINT " ";: GOTO 100
300 SOUND 1,2000,2
310 LOCATE M,K+1 : PRINT " ";
320 IF MF > 50 THEN 350
330 GOTO 130
340 '
350 ' FIM DO JOGO
360 '
370 CLS
380 LOCATE 12,12
390 PRINT "Você pontuou:";SC
400 SOUND 1,180,100
410 END
```

Na bolsa

É chegado o momento de despir o fato de treino e vestir-se a rigor. Hoje você vai até à bolsa negociar uns títulos.

Você dispõe de 1000 libras com as quais deve comprar acções. O Amstrad fará todas as perguntas necessárias. Tenha cuidado, não compre mais acções do que aquelas que pode pagar; o computador não o deixará fazer batota e impedirá qualquer atitude ilegal.

```
10 ' NA BOLSA
20 '
30 CLS
40 PRINT
50 PRINT TAB(10);" NA BOLSA."
60 WHILE INKEY$=" " : WEND
70 SOUND 1,180
80 X=4 : I=10
90 B=100
100 DIM A(3,4)
110 FOR A=1 TO X
120 A(1,A)=A+INT (RND*250)
130 A(3,A)=0
140 NEXT
150 CLS
160 PRINT
170 PRINT "NO. ACÇÕES PREÇO ORIGINAL "
180 PRINT " ==========================
==================="
190 FOR A=1 TO X
200 PRINT
210 Z=A(1,A)-A(3,A)
220 PRINT A;TAB (X+2);A(2,A);TAB(16);A(1,A);
    TAB(27);A(3,A);TAB(34);Z
230 NEXT
240 PRINT : PRINT
250 PRINT "BANCO";B;" TAXA DE JURO:";I
260 PRINT
270 PRINT "QUE EMPRESA (1-4)";A
280 IF A > 4 THEN PRINT "A EMPRESA NÃO ESTÁ A SER
    NEGOCIADA": PRINT : GOTO 270
290 PRINT : PRINT
300 INPUT "COMPRA OU VENDA";A$
310 PRINT : PRINT
320 INPUT "QUANTAS ACÇÕES";C
```

```
330 IF LEFT$(A$,1)="S" THEN C=-C
340 IF C > 0 THEN A (3,A)=A(1,A)
350 B=B-C*A(1,A)
360 IF B< =0 THEN 500
370 A(2,A)=A(2,A)+C
380 IF A(2,A) < 0 THEN 520
390 FOR A=1 TO X
400 A(1,A)=A(1,A)+INT(RND*(A(1,A)/4))-INT
    (RND*(A(1,A)/4))
410 IF A(1,A) < 1 THEN A(1,A)=1
420 NEXT
430 I=INT(ABS(I+RND*8-X)+1)
440 B=INT(B+B*I/100)
450 IF RND > 0.05 THEN 150
460 PRINT "COLAPSO"
470 P=A(2,1)+A(2,2)+A(2,3)-1000
480 PRINT "VALOR:";P
490 END
500 PRINT "BANCARROTA"
510 END
520 PRINT "FRAUDE"
530 END
```

Campo de batalha

O labirinto de Munchman, onde este normalmente reside, transformou-se no centro de uma batalha entre os galácticos e os subidores de plataformas. O seu labirinto é agora uma barafunda, e existem neste momento menos dois fantasmas do que é habitual; os outros dois, além das pílulas de energia em falta, são vítimas do conflito que se prepara. No entanto, tudo parece na mesma para o guloso comedor de pílulas.

O leitor controla Munchman com as teclas A (para cima), Z (para baixo), a tecla da vírgula (para a esquerda) e a de ponto final (para a direita). Note que os fantasmas parecem aproximar-se sempre da sua posição. Não é um jogo extremamente rápido, mas tentámos criar um compromisso entre a inteligência dos fantasmas e a velocidade global, a fim de conseguirmos uma boa aparência do jogo no visor.

```
10 ' CAMPO DE BATALHA
20 '
30 DEFINT A-Z
40 S=0 : LV=3
50 GOSUB 790
```

```
60 GOSUB 660
70 LOCATE A,B+1 : PEN 1 : PRINT A$
80 '
90 ' CICLO PRINCIPAL
100 '
110 GOSUB 170
120 GOSUB 170
130 GOSUB 360
140 IF S > 6000 THEN GOSUB 820 : GOSUB 660 : GOTO 70
150 GOTO 90
160 '
170 ' ACEITAR A ENTRADA DO JOGADOR E MOVER
180 '
190 IF (X(1)=A AND Y(1)=B) OR (X(2)=A AND Y(2)=B) THEN
    570
200 LOCATE 7,1 : PRINT S;
210 LOCATE A,B+1 : PRINT " ";
220 D$=INKEY$ : IF D$ = " " THEN 270
230 IF D$=","  THEN A$="e" : C=-1 : D=0
240 IF D$="."  THEN A$="c" : C=1 : D=0
250 IF D$="A" THEN A$="b" : D=-1 : C=0
260 IF D$="Z" THEN A$="d" : D=1 : C=0
270 IF (C=-1 AND P(A-1,B)=1) OR (C=1 AND P(A+1,B)=1)
    OR (D=-1 AND P(A,B-1)=1) OR (D=1 AND P(A,B+1)=1)
    THEN 290
280 A=A+C : B=B+D
290 LOCATE A,B+1 : PRINT " ";
300 LOCATE A,B+1
310 PEN 1
320 PRINT A$
330 IF P(A,B)=2 THEN P(A,B)=0 : LOCATE A,B+1 : S=S+10:
    SOUND 1,300,1
340 RETURN
350 '
360 ' MOVER FANTASMAS E VERIFICAR PERDA
370 '
380 FOR T=1 TO 2
390 LOCATE X(T), Y(T)+1
400 PEN 2
410 IF P(X(T),Y(T))=2 THEN PRINT ".": ELSE PRINT " ";
420 PEN 1
430 IF X(T)=A AND Y(T)=B THEN 570
440 IF X(T) > A AND P(X(T)-1,Y(T))< >1 THEN X(T)=X(T)-1
450 IF X(T)< A AND P(X(T)+1,Y(T))< >1 THEN X(T)=X(T)+1
```

```
460 IF Y(T) > B+1 AND P(X(T), Y(T)-1< >1 THEN Y(T)=
    Y(T)-1
470 IF Y(T)< B+1 AND P(X(T),Y(T)+1)<>1 THEN Y(T)=Y(T)+1
480 IF X(2)=X(1) AND Y(2)=Y(1) THEN K=INT (RND*2) -INT
    (RND*2) : L = INT(RND*2-INT(RND*2) : IF) P(X(1)+K,Y
    (1)+L)< >P THEN X(1)=X(1)+K : Y(1)=Y(1)+L
490 LOCATE X(T),Y(T)+1
500 PEN 3
510 PRINT ''f'';
520 PEN 1
530 IF RND > 0.94 THEN LOCATE X(2),Y(2)+1 : PRINT '' '';:
    X(2)=INT(RND*30)+5 : Y(2)=INT(RND*20)+3 : IF P(X(2),
    Y(2))=1 OR A=X(2) OR B=Y(2) THEN 530
540 NEXT
550 RETURN
560 '
570 ' COMER
580 '
590 SOUND 1,800,40
600 LV=LV-1
610 IF LV=0 THEN CLS : LOCATE 12, 12 : PRINT ''PONTUA-
    ÇÃO:''; S : END
620 GOSUB 1430
630 GOSUB 660
640 GOTO 70
650 '
660 ' DESENHAR VISOR
670 '
680 MODE 1
690 PEN 1
700 CLS : PRINT ''PONTUAÇÃO: '';S;''MUNCHMAN C.1984''
710 PEN 2
720 FOR Y=1 TO 24 : FOR X=1 TO 40
730 LOCATE X,Y+1
740 IF P(X,Y)=1 THEN PRINT CHR$(143);
750 IF P(X,Y)=2 THEN PRINT CHR$ (46);
760 NEXT : NEXT
770 RETURN
780 '
790 ' INICIALIZAÇÃO
800 '
810 DIM P(40,24)
820 FOR Y=1 TO 24 : FOR X=1 TO 40
830 READ P(X,Y)
```

```
840 NEXT : NEXT
850 DATA 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
860 DATA 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
870 DATA 1,2,2,2,2,2,1,2,2,2,2,1,2,2,2,2,2,2,1
880 DATA 1,1,2,2,2,2,1,1,1,2,2,1,2,2,2,2,2,2,1
890 DATA 1,2,2,2,1,2,2,2,1,2,2,2,2,2,2,2,2,1,2,1
900 DATA 2,2,2,2,2,2,1,1,2,2,2,2,2,2,2,2,2,2,1
910 DATA 1,2,2,2,2,2,2,1,1,1,2,2,2,2,2,1,2,2,2,2
920 DATA 2,2,1,1,1,1,1,1,2,2,1,1,1,1,1,1,1,1,2,1
930 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
940 DATA 2,2,1,1,1,1,1,2,2,2,2,1,2,2,1,2,2,1,2,1
950 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
960 DATA 2,2,2,2,1,1,1,1,1,2,2,2,2,1,2,2,1,2,2,1
970 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
980 DATA 1,2,2,2,1,1,1,1,1,2,2,2,2,2,2,2,1,1,2,1
990 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1000 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1010 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1020 DATA 2,2,2,2,1,1,1,2,2,2,2,2,2,2,2,2,2,2,1
1030 DATA 1,2,2,2,2,1,1,2,2,1,2,2,2,2,2,2,2,1,2,2
1040 DATA 2,2,2,2,2,2,1,1,1,2,2,2,1,2,1,2,2,2,2,1
1050 DATA 1,2,2,2,2,2,2,1,1,2,2,2,2,1,2,2,2,2,2,2
1060 DATA 2,2,2,2,2,1,1,1,2,2,2,2,1,1,1,2,2,2,2,1
1070 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1080 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1090 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1100 DATA 2,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1110 DATA 1,2,2,2,2,2,2,1,1,1,2,2,1,1,2,2,2,2,2,1
1120 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1130 DATA 1,2,1,1,2,2,1,1,2,1,1,1,1,1,2,1,1,2,1,1
1140 DATA 1,1,1,2,2,2,2,2,2,1,2,1,2,2,2,2,2,2,2,1
1150 DATA 1,2,2,2,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2
1160 DATA 1,2,2,2,1,2,2,1,1,2,2,2,1,2,2,1,2,2,2,1
1170 DATA 1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1180 DATA 2,2,2,2,1,2,1,2,2,1,2,2,2,1,1,2,2,2,2,1
1190 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1200 DATA 1,1,1,1,1,2,2,2,1,2,2,2,2,2,2,2,2,2,1
1210 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1220 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1230 DATA 1,2,2,2,2,2,1,2,2,2,2,2,1,1,1,2,1,1,1,2
1240 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1250 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,1
1260 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1270 DATA 1,2,2,2,1,2,2,2,2,2,2,2,2,2,2,1,2,2,1
```

```
1280 DATA 1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1
1290 DATA 1,2,2,2,1,2,2,2,2,2,2,2,2,2,2,2,2,1,1
1300 DATA 1,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,1,1,2,1
1310 DATA 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1320 DATA 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
1330 S=0
1340 BORDER 0 : INK 0,0 : INK 2,2 : INK 1,24
1350 A=18 : B=14 : C=0 : D=0 : E=A : F=B
1360 SYMBOL AFTER 96
1370 SYMBOL 97,60,126,255,255,255,255,126,60
1380 SYMBOL 98,0,66,195,231,255,255,126,60
1390 SYMBOL 99,60,126,248,240,240,248,126,60
1400 SYMBOL 100,60,126,255,255,231,195,66,0
1410 SYMBOL 101,60,126,31,15,15,31,126,60
1420 SYMBOL 102,28,62,42,107,127,127,109,73
1430 A=18 : B=14 : C=0 : D=0 : E=A : F=B
1440 A$="e"
1450 RESTORE 1490
1460 FOR T=1 TO 2
1470 READ X(T),Y(T)
1480 NEXT
1490 DATA 4,3,35,22
1500 RETURN
```

Letras

Volte à sua meninice... Recorda-se certamente de brincar com pequenos quadrados cada um com uma letra. Havia sempre um espaço entre os quadrados que o leitor tinha de rodar, tentando criar palavras ou colocar as letras por ordem alfabética. Bem, este programa recria esse jogo.

Nesta versão computadorizada, o leitor deve colocar as peças por ordem alfabética, deixando o espaço livre no canto inferior direito. Deve indicar as coordenadas da peça que deseja mover e do ponto para onde deseja movê-la; as coordenadas são apresentadas no lado direito do visor. Os movimentos ilegais são imediatamente rejeitados.

O jogo não é tão simples como parece — dê algum trabalho ao seu cérebro, em vez de exercitar apenas os dedos como nos jogos anteriores!

```
10 ' LETRAS
20 '
30 GOSUB 390
40 GOSUB 190
50 GOSUB 190
```

```
60 SOUND 1,180 : SOUND 1,240 : SOUND 1,120
70 PRINT : PRINT : PRINT
80 PEN 2
90 PRINT ''QUE PEÇA DESEJA DESLOCAR?''
100 INPUT X
110 IF A (X)=32 THEN 100
120 INPUT ''PARA ONDE'';Y
130 IF A(Y)<>32 THEN 120
140 A(Y)=A(X)
150 A(X)=32
160 GO=GO+1
170 GOTO 50
180 '
190 ' IMPRIMIR
200 '
210 PEN 2
220 CLS
230 PRINT TAB(14); ''L E T R A S''
240 PRINT TAB(13); ''=============''
250 PRINT : PRINT
260 PRINT ''DESLOCAR NÚMERO:'';GO
270 PRINT : PRINT
280 L$=CHR$(143)+CHR$(143)+CHR$(143)+CHR$(143)
    +CHR$(143)+CHR$(143)+CHR$(143)+CHR$(143)
    +CHR$(143)+CHR$(143)+CHR$(143)
290 PEN 3
300 PRINT TAB(8);L$
310 PRINT TAB(8);CHR$(143);: PEN 1 : PRINT '' ''; CHR$(A(1));
    '' '';CHR$(A(2));'' '';CHR$(A(3));'' '';CHR$(A(4));'' '';:
    PEN 3 : PRINT CHR$(143); ''1 2 3 4''
320 PRINT TAB(8); CHR$(143);'' '';: PEN 1 : PRINT CHR$(A(5));
    '' '';CHR$(A(6));'' '';CHR$(A(7));'' ''; CHR$(A(8));'' '';:
    PEN 3 : PRINT CHR$(143); ''5 6 7 8''
330 PRINT TAB(8);CHR$(143);'' '';: PEN 1 : PRINT CHR$(A(9));
    '' '';CHR$(A(10));'' '';CHR$(A(11));'' '';CHR$(A(12));'' '';:
    PEN 3 : PRINT CHR$(143); ''9 10 11 12''
340 PRINT TAB(8); CHR$(143);'' ''; : PEN 1 : PRINT CHR$
    (A(13));'' '';CHR$(A(14));'' '';CHR$(A(15));'' '';CHR$(A(16));
    '' '';: PEN 3 : PRINT CHR$(143);''13 14 15 16''
350 PEN 3 : PRINT TAB(8);L$
360 RETURN
370 PEN 2
380 '
390 ' INICIALIZAR
400 '
```

```
410 DIM A(16)
420 FOR B=1 TO 16
430 READ M
440 A(B)=M+64
450 NEXT
460 GO=1
470 RETURN
480 DATA 9,14,5,2,11,6,1,4,12
490 DATA 7,-32,10,13,8,3,15
```

Outro tijolo na parede

O leitor deverá deslocar o seu «vampiro» para a esquerda e a direita usando as teclas 7 e 9 do teclado numérico (ficará surpreendido com a facilidade de uso destas duas teclas — um facto descoberto após muitas horas de teste de jogos!).

```
10 ' OUTRO TIJOLO NA PAREDE
20 '
30 H=0
40 INK 0,0 : INK 1,22 : INK 3,3
50 INK 2,2 : INK 3,6
60 BORDER 0
70 SPEED KEY 1,2
80 GOSUB 660
90 SPEED KEY 1,2
100 FOR Z=1 TO 5
110 A$=A$+A$
120 NEXT Z
130 PEN 3 : CLS
140 LOCATE 3,1
150 PRINT A$
160 LOCATE 3,2
170 PRINT A$
180 LOCATE 3,3
190 PRINT A$
200 G$=A$ : LOCATE 3,22 : PRINT G$
210 PEN 1
220 LOCATE 8,25
230 PEN 3
240 FOR X=1 TO 22
250 LOCATE 2,X
260 PRINT ''d''
```

```
270 LOCATE 35,X
280 PRINT "d";
290 NEXT X
300 IF LV=0 THEN 570
310 FOR T=1 TO 1500 : NEXT : SOUND 1,100
320 P$=INKEY$
330 IF P$="9" AND A< 30 THEN A=A+2
340 IF P$="7" AND A > 2 THEN A=A-2
350 IF A=2 THEN LOCATE A,21 : PEN 3 : PRINT "d"; : PEN
    2 : PRINT "abc" : GOTO 380
360 IF A=30 THEN LOCATE A,21 : PRINT "abc";: PEN 3 :
    PRINT "d" : PEN 2 : GOTO 380
370 LOCATE A,21 : PEN 2 : PRINT "abc"
380 LOCATE D,C : PRINT " "
390 IF C   21 THEN LV=LV-1 : SOUND 1,400 : C = 6 : D =
    INT (RND*25) + 10 : LOCATE 2,22 : PEN 3 : PRINT G$+
    "d" : PEN 1 : GOTO 300
400 IF D > 33 OR D< 4 THEN G=-G : SOUND 1,140,15
410 C=C+F
420 D=D+G
430 LOCATE D,C : PEN 1 : PRINT B$ : PEN 2
440 IF C < 4 THEN GOSUB 490
450 IF C<>21 THEN GOTO 470
460 IF D=A+2  OR D=A+3  OR D=A+4 THEN F=-F
    :SOUND 1,100,15
680 F=1 : G=1 : C=6
690 D=10+INT(RND*25) : A$="d"
700 B$ = CHR$(231)
710 REM +++ DEFINIR GRÁFICOS +++
720 SYMBOL AFTER 95
730 SYMBOL 97,1,31,127,255,255,255,255,248
740 SYMBOL 98,255,255,255,255,255,255,0,0
750 SYMBOL 99,128,248,255,255,255,255,31
760 SYMBOL 100,254,254,254,0,239,239,239,0
770 REM +++ a b c = ESCUDO +++
780 REM +++   d   = TIJOLOS +++
790 RETURN
```

Cavaleiros

Apresentamos agora uma variante do xadrez/damas que certamente agradará ao leitor.

Este jogo é jogado num tabuleiro de sete por sete quadrados. O Amstrad indica-nos qual a peça a deslocar; cada peça é indicada escrevendo

um número de dois algarismos, sendo o primeiro a coordenada do lado do tabuleiro e o segundo a coordenada no topo. É necessário então decidir para onde deve ser realizado o movimento. Todas as peças se deslocam como os cavalos do xadrez.

Captura-se uma peça inimiga «aterrando» sobre ela. O primeiro jogador que capture cinco das peças inimigas é o vencedor. O Amstrad é um forte oponente, rápido e habilidoso — o leitor necessitará de toda a sua arte para o vencer!

```
10 ' CAVALEIROS
20 '
30 GOSUB 840
40 GOSUB 550
50 IF HU=5 OR CO=5 THEN 750
60 GOSUB 340
70 GOSUB 550
80 IF HU=5 OR CO=5 THEN 750
90 GOSUB 120
100 GOTO 40
110 '
120 ' MOVIMENTOS DO JOGADOR
130 '
140 Q=0
150 M=INT (RND*66)+11
160 Q=Q+1
170 IF Q=500 THEN 750
180 IF H(M)< >72 THEN 150
190 PRINT ''DEVE DESLOCAR A PEÇA EM '';M
200 INPUT N
210 IF N=99 THEN Q=500 : GOTO 750
220 '
230 ' MOVIMENTO LEGAL?
240 '
250 P=0
260 CT=1
270 IF M+Z(CT)=N THEN P=1
280 IF CT< 8 THEN CT=CT+1 : GOTO 270
290 IF P=0 THEN PRINT ''MOVIMENTO ILEGAL'' : GOTO 200
300 IF H(N)=67 THEN HU=HU+1 : PRINT ''BEM JOGADO!''
    : FOR R=1 TO 800 : NEXT
310 H(M)=46 : H(N)=72
320 RETURN
330 '
340 ' MOVIMENTOS DO COMPUTADOR
```

```
350 '
360 Q1=0
370 Q1=Q1+1
380 K=INT (RND*66)+11
390 IF Q1=500 THEN 750
400 IF H(K)< >67 THEN 370
410 PRINT ''DEVO DESLOCAR A PEÇA EM '';K
420 W=1
430 FOR T=1 TO 1800 : NEXT
440 IF K+Z(W)< 11 OR K+Z(W) > 77 THEN 460
450 IF H(K+Z(W))=72 THEN PRINT ''CONSEGUI!!'' : CO=
    CO+1 : FOR P=1 TO 300 : NEXT : GOTO 510
460 IF W< 8 THEN W=W+1 : GOTO 440
470 W=1
480 IF (K+Z(W)< 11 OR K+Z(W) > 77) AND W < 8 THEN
    W=W+1 : GOTO 480
490 IF H(K+Z(W))< >46 AND W < 8 THEN W=W+1 : GOTO
    490
500 IF W=8 AND H(K+Z(W))< >46 THEN Q1=500 : GOTO 750
510 X=K : Y=K+Z(W)
520 H(X)=46 : H(Y)=67
530 RETURN
540 '
550 ' IMPRIMIR TABULEIRO
560 '
570 CLS : PRINT : PRINT : PRINT
580 PRINT TAB(8); ''A MINHA PONTUAÇÃO É''; CO
590 PRINT TAB(7); ''E A SUA É''; HU
600 PRINT
610 PRINT TAB(8); ''1 2 3 4 5 6 7''
620 PRINT TAB(8); ''———————————''
630 FOR M=70 TO 10 STEP −10
640 PRINT TAB(5);M/10;
650 FOR N=1 TO 7
660 PRINT CHR$(H(M+N));'' '';
670 NEXT
680 PRINT M/10 : PRINT
690 NEXT
700 PRINT TAB(8); ''———————————''
710 PRINT TAB(8); ''1 2 3 4 5 6 7''
720 PRINT
730 RETURN
740 '
750 ' FIM DO JOGO
```

```
760 '
770 GOSUB 550
780 IF HU=5 THEN PRINT ''GANHOU, HUMANO''
790 IF CO=5 THEN ''ESSA VITÓRIA É O MEU PRIMEIRO
    PASSO PARA O DOMÍNIO DO MUNDO!''
800 IF Q=500 THEN PRINT ''ACEITO O SEU DESEJO DE
    DESISTIR''
810 IF Q1=500 THEN PRINT ''DESISTO FRENTE A UM
    MESTRE!''
820 END
830 '
840 ' INICIALIZAR
850 '
860 CLS : PRINT ''POR FAVOR PREPARE-SE...''
870 DIM H(99),Z(8)
880 X=0 : Q1=0 : Q=0
890 RANDOMIZE TIME
900 HU=0 : CO=0
910 FOR A=1 TO 99
920 IF A > 77 OR A=70 OR A=60 OR A=68 OR A=69 OR
    A=50 OR A=59 OR A=40 OR A=48 OR A=49 THEN 970
930 IF A=30 OR A=38 OR A=39 OR A=20 OR A=28 OR
    A=29 OR A=18 OR A=19 OR A    THEN 970
940 H(A)=46
950 IF A > 70 AND A < 78 THEN H(A)=67
960 IF A > 10 AND A < 18 THEN H(A)=72
970 NEXT
980 FOR A=1 TO 8 : READ Z(A) : NEXT
990 DATA -8,-21,-12,-19,19,12,21,8
1000 RETURN
```

14

TRATAMENTO DE TEXTO

Máquina de escrever

Um dos programas mais úteis existentes tanto para a actividade comercial como para os utilizadores domésticos é o processador de texto. O *Amsword*, a versão para o Amstrad do *Tasword 2* da Tasman, é reconhecido como um soberbo tratamento de texto.

No entanto, muitos utilizadores apenas pretendem um programa que permita usar o computador como máquina de escrever, corrigir rapidamente os erros, imprimir automaticamente endereços e enviar a carta formatada para a impressora.

Máquina de escrever, o programa que se segue, é uma simples tentativa de construção de um programa deste tipo e fácil de usar. As diversas secções do programa estão cuidadosamente referenciadas, de modo a permitir ao leitor alterá-las de acordo com os seus requisitos próprios. Pode certamente constituir a base de sistemas mais complexos com justificação à direita, contagem de palavras, e muitas outras facilidades.

O uso do programa é bastante fácil. Escreva até encher uma linha no visor (o que equivale a cerca de 66 caracteres) e carregue em seguida na tecla ENTER; a primeira linha da sua carta passa assim a estar registada na memória do computador. O Amstrad poderá guardar até 60 linhas, mas o leitor poderá alterar este número se as suas necessidades forem diferentes. Convirá notar que, no caso de uma linha exceder 66 caracteres quando é introduzida na máquina, é automaticamente apagada e o leitor deverá escrevê-la de novo. São fornecidas instruções no visor, indicando os controlos a usar para gravar uma carta em fita, para a recuperar desta, para iniciar uma carta nova ou imprimir uma já terminada.

Um último aspecto. Nas linhas 280-310 aparece um endereço fictício — este deve ser substituído pelo seu próprio endereço, escrito do mesmo modo.

```
10 ' MÁQUINA DE ESCREVER .2
20 '
30 K=0 : T=1
40 MODE 2
50 INK 0,0
60 PRINT : PRINT : PRINT ''TECLA 2: VERDE; QUALQUER
   TECLA, BRANCO''
70 A$=INKEY$ : IF A$= '' '' THEN /0
80 IF A$=''2'' THEN INK 1,18 ELSE INK 1,26
90 PRINT : PRINT : PRINT ''DATA, POR FAVOR'' : LINE
    INPUT D$
100 CLS
110 GOSUB 840
120 DIM W$(62)
130 LOCATE 1,1
140 FOR T=1 TO 60
150 LINE INPUT W$(T) : IF LEN (W$(T)) > 60 THEN SOUND
    1,100,9 : LOCATE 1,T : PRINT '' | '';: GOSUB 840 : W$
    (T)='' '' : GOTO 150
160 IF W$(T)=''  '' THEN K=T : T=60
170 IF W$(T)=''  '' THEN CLS : GOSUB 840 : GOTO 130
180 IF W$(T)=''/'' THEN 400
190 IF W$(T)=''] '' THEN 560
200 IF T=40 THEN GOSUB 840
210 NEXT T
220 PRINT # 8,CHR$(27);CHR$(69)
230 PRINT # 8,CHR$(27);CHR$(65);CHR$(16)
240 WIDTH 66
250 '
260 ' NOTAR QUE PARA MUDAR ENDEREÇO BASTA ESCRE-
    VER O ENDEREÇO NOVO NAS INSTRUÇÕES DATA
270 '
280 PRINT # 8,TAB(45);''10, AV. LIBERDADE''
290 PRINT # 8,TAB(45);''1000 LISBOA''
300 PRINT # 8,TAB(45);''PORTUGAL''
320 PRINT# 8,'' ''
330 PRINT# 8,TAB(42);D$
340 PRINT# 8, '' ''
350 FOR T=1 TO K-1
360 PRINT# 8, '' '';W$(T)
370 NEXT T
380 END
390 '
400 ' GRAVAR FICHEIRO
```

```
410 '
420 CLS
430 LOCATE 27,8
440 PRINT ''INDIQUE NOME (ATÉ 14 LETRAS)''
450 PRINT : PRINT
460 INPUT FL$
470 PRINT : PRINT
480 OPENOUT FL$
490 FOR T=1 TO 60
500 PRINT # 9,W$(T)
510 NEXT
520 PRINT # 9,D$
530 CLOSEOUT
540 CLS : GOSUB 840 : GOTO 130
550 '
560 ' CARREGAR FICHEIRO
570 '
580 CLS
590 LOCATE 30,4
600 PRINT ''INDIQUE NOME''
610 PRINT : PRINT
620 INPUT FG$
630 PRINT : PRINT : PRINT
640 OPENIN FG$
650 FOR T=1 TO 60
660 INPUT # 9,W$(T)
670 NEXT
680 INPUT # 9,D$
690 IF EOF-1 THEN CLOSEIN
700 CLOSEIN
710 CLS
720 GOSUB 840
730 LOCATE 1,1
740 FOR T=1 TO 60
750 PRINT W$(T)
760 WHILE INKEY$=" " : WEND
770 NEXT
780 GOSUB 840
790 T=61 : INPUT ''CONTINUE'';W$(T)
800 GOTO 160
810 '
820 ' IMPRIMIR MENU
830 '
840 LOCATE 61,1
```

```
850 PRINT ''—— MÁQUINA DE ESCREVER——'';
860 FOR G=2 TO 24 : LOCATE 61,G : PRINT ''|'';: NEXT G
870 LOCATE 64,8 : PRINT '' ' \ '=imprimir''
880 LOCATE 64,12 : PRINT'' '  '=recomeçar''
890 LOCATE 64,16 : PRINT '' '/'=gravar ficheiro''
900 LOCATE 64,20 : PRINT '' ']'=carregar ficheiro''
910 LOCATE 1,T
920 RETURN
```

15

UMA EXPLICAÇÃO GRÁFICA

Nos últimos anos, os maiores desenvolvimentos no campo dos computadores domésticos ocorreram certamente ao nível das suas capacidades gráficas. Há quatro ou cinco anos, os computadores domésticos baratos possuíam apenas baixa resolução, equivalendo à dos próprios caracteres impressos. Para conseguirem uma simples resolução média de 60 por 100 pontos (*pixels*), os possuidores de micros viam-se muitas vezes forçados a recorrerem a periféricos caros.

O seu Amstrad possui já excelentes qualidades gráficas: 27 cores, uma resolução máxima de 640 por 200, comandos para desenho de linhas, marcar formas complexas, verificar a cor de uma certa posição no visor, realizar animação, e muito mais. E o aspecto interessante é que o uso destas facilidades não diminui significativamente a memória disponível.

Neste capítulo estudaremos a gama de facilidades gráficas disponíveis no Amstrad, explicando-as separadamente. O capítulo encontra-se dividido em três secções principais — os comandos que tratam a cor no caso da impressão de texto, os que têm a ver com a criação de gráficos pelo utilizador e os que controlam os gráficos de alta resolução. Em cada secção apresentam-se vários programas, alguns apenas para demonstração, mas verificará que muitos são bastante úteis, particularmente quando incorporados nos seus próprios programas.

Cor, modos e comandos de texto

O leitor terá já compreendido que o Amstrad possui vários modos gráficos que têm a ver com o número de caracteres que se podem imprimir no visor, o número de cores utilizáveis e o nível de resolução gráfica.

O modo 1 é o mais usado e permite o emprego de 40 caracteres por linha quando se processa texto, até quatro cores e uma resolução gráfica de 320 por 200.

O modo 2 é mais interessante para trabalho comercial, dado que permite o emprego de 80 caracteres por linha (o que o torna ideal para uso em CP/M — disponível em disco para o Amstrad). São apenas permitidas duas cores — uma de fundo e outra de primeiro plano, mas pode-se usar a maior resolução possível no Amstrad, 640 por 200 *pixels*.

O modo 0 permite 20 caracteres por linha e uma resolução gráfica máxima de 160 por 200. No entanto, a sua grande vantagem é que permite o emprego de 16 cores no visor em qualquer momento.

Estes três modos são acedidos utilizando o comando MODE, seguido pelo número 0, 1 ou 2.

O leitor estará um pouco confuso talvez com a expressão «alta resolução». Mais adiante, neste capítulo, estudaremos os vários comandos que controlam o uso dos gráficos de alta resolução, mas por agora convirá já indicar o que significa esta expressão e o termo *pixels*.

Se observar cuidadosamente um caracter impresso no visor, verificará que é formado por um certo número de pontos. Cada um destes é designado por *pixel*, e a cada *pixel* corresponde uma localização fixa no visor. As dimensões dadas quando se fala de resolução indicam o número de *pixels* existentes no visor, primeiro na horizontal e depois na vertical.

Como cada caracter é constituído por um certo número de *pixels*, podemos dizer que os comandos de alta resolução nos dão acesso aos *pixels* individuais, enquanto os comandos de texto apenas nos permitem alterar caracteres e as respectivas posições.

Observemos o texto no visor, e vejamos como os caracteres são impressos nele. Muitos dos caracteres disponíveis podem ser encontrados simplesmente carregando numa tecla; no entanto, existem vários caracteres que não podem ser acedidos directamente pelo teclado e que devem ser impressos usando o comando CHR$. Se se imprime CHR$(n), onde n é o código do caracter entre 32 e 128, obtém-se um caracter no visor. Por exemplo, se imprimirmos CHR$(224), obteremos uma face sorridente (nas costas do manual do Amstrad pode-se encontrar uma lista de todos os caracteres disponíveis).

Já discutimos a instrução PRINT e vimos como é possível usar TAB para deslocar a posição de impressão um dado número de colunas; observámos ainda como se pode usar o ponto e vírgula como separador. Vejamos agora melhor como actua a instrução LOCATE.

Se o leitor já utilizou outros computadores, conhece um comando que permite imprimir numa certa coluna e linha, indicadas pelas respectivas coordenadas. As versões desta instrução existentes em muitos computadores são designadas por PRINT AT ou PRINT @, mas no Amstrad chama-se LOCATE.

LOCATE é seguida de dois números, que correspondem às coordenadas X e Y da posição de impressão desejada. Vejamos um exemplo do uso de LOCATE:

```
10 ' LOCATE, LOCATE
20 '
30 CLS
40 FOR Y=1 TO 25 : ' COORDENADA Y
50 FOR X=1 TO 39 STEP 3 : ' COORDENADA X
60 LOCATE X,Y
70 PRINT STR$(X+1);
80 NEXT : NEXT
90 GOTO 90
```

São usadas duas instruções para determinar a posição do cursor. Trata-se de VPOS e POS, e vejamos como actuam:

```
10 ' POS/VPOS CRIA UM QUARTO DE VISOR
20 '
30 CLS
40 FOR T=1 TO 3000
50 A$=INKEY$: IF A$=" " THEN 50
60 IF POS (#0) > 20 THEN PRINT : GOTO 90
70 IF VPOS (#0) > 12 THEN END
80 PRINT A$;
90 NEXT
```

O programa anterior utiliza VPOS e POS para criar um minivisor, quase como uma janela. Qualquer que seja a tecla premida, esta é apresentada no visor e, quando se tem mais de 20 caracteres numa linha, o texto passa para a linha seguinte (como acontece no visor verdadeiro). Depois de preenchidas doze linhas, o programa pára.

Note-se o 0 entre parêntesis tanto em VPOS como em POS. Este zero refere-se ao tipo de *stream* (discutido no Capítulo 19). Trata-se do código de *stream* do visor inteiro, mas POS e VPOS podem também ser usadas para verificar a posição de janelas e cursores de impressora, etc; POS e VPOS são comandos muito úteis para uso em programas como os processadores de texto. O valor # N entre parêntesis, onde N é um número de *stream* entre zero e nove, deve ser colocado a seguir a qualquer instrução VPOS ou POS.

Como o leitor já terá certamente deduzido, VPOS trata a posição vertical do cursor, enquanto POS trata a posição horizontal. Na secção relativa aos gráficos de alta resolução, o leitor encontrará duas instruções semelhantes a estas, XPOS e YPOS.

Um arco-íris

O seu Amstrad dispõe de grandes facilidades de tratamento da cor — usando um conjunto de 27 cores, que podem ser utilizadas em fundo,

em primeiro plano e na margem do visor. Muitos utilizadores de outras máquinas ficarão satisfeitos por poderem usar cores como o cinzento e o castanho, muito úteis em jogos e imagens.

Voltando à nossa discussão dos três modos do visor, o leitor recorda que existe um limite para o número de cores diferentes que se pode ter no visor em qualquer momento. Este número varia com o modo utilizado, existindo um compromisso entre o nível de resolução e o número de cores permitido.

A forma mais simples de observar estas cores consiste em usar o comando BORDER. Trata-se de uma instrução muito simples, que define a cor da margem do visor; o número que se segue ao comando determina a cor usada. A tabela 1 mostra as cores disponíveis e os respectivos códigos.

TABELA 1

CORES DO AMSTRAD E RESPECTIVOS CÓDIGOS

Código da cor	Cor
0	Negro
1	Azul-escuro
2	Azul-claro
3	Vermelho
4	Magenta
5	Malva
6	Vermelho-claro
7	Púrpura
8	Magenta-claro
9	Verde
10	Cião
11	Azul-celeste
12	Amarelo
13	Branco-cinzento
14	Azul-pastel
15	Laranja
16	Rosa
17	Magenta-pastel
18	Verde-claro
19	Verde-marítimo
20	Cião-claro
21	Verde-amarelado
22	Verde-pastel
23	Cião-pastel
24	Amarelo-claro
25	Amarelo-pastel
26	Branco-brilhante

O curto programa que se segue imprime uma margem de cor diferente sempre que se carrega numa tecla:

```
10 ' MARGEM COLORIDA
20 '
30 FOR T=0 TO 26
40 BORDER T
50 WHILE INKEY$=" ": WEND
60 SOUND 1,400-(12*T)
70 NEXT
```

Pode-se fazer cintilar a margem colocando dois números após a instrução BORDER — a cor variará entre as duas indicadas. Escreva algumas instruções BORDER seguidas de dois números a fim de observar o efeito criado. Se não gostar da velocidade da troca de cores, experimente a instrução SPEED INK. O formato desta é:

```
SPEED INK A,B
```

Tanto A como B são números de imagens (são enviadas para o visor 50 imagens por segundo) e determinam o tempo durante o qual a margem manterá a mesma cor. Assim, se quisermos que a primeira cor se mantenha durante meio segundo e a segunda durante um segundo e meio, escreveremos a seguir à instrução SPEED INK os números 25 e 75. Vejamos um programa de demonstração que incorpore várias instruções SPEED INK e BORDER. Experimente por si mesmo e veja os efeitos que consegue criar.

```
10 ' MARGENS
20 '
30 GOSUB 160
40 BORDER INT (RND*27),1
50 FOR T=1 TO 3500 : NEXT
60 GOSUB 160
70 SPEED INK 5,5
80 BORDER 6,4
90 FOR T=1 TO 3500 : NEXT
100 GOSUB 160
110 SPEED INK 80,10
120 BORDER 24,13
130 FOR T=1 TO 5000 : NEXT
140 GOSUB 160
150 END
160 ' RECOMEÇO E PAUSA
```

```
170 BORDER 1
180 SOUND 1,100,20,15
190 SPEED INK 20,20
200 FOR T=1 TO 1000 : NEXT
210 RETURN
```

O comando SPEED INK pode ser igualmente usado para modificar a instrução que veremos em seguida — o versátil comando INK.

Nos capítulos anteriores vimos que podemos modificar a cor usada para primeiro plano e fundo usando as instruções PEN e PAPER, respectivamente. Em modo 1, por exemplo, vimos como é possível alterar qualquer das cores: azul-escuro, azul-claro, amarelo ou vermelho.

A instrução INK permite-nos mudar as cores que podem ser usadas pelos comandos PEN e PAPER. Para tornar isto mais claro, consideremos uma paleta de quatro cores. Se pensarmos no comando INK como uma caixa de diferentes tintas, basta-nos especificar as que desejamos usar. A seguir ao comando INK usam-se pelo menos dois números, correspondendo o primeiro à cor PEN/PAPER; PEN 0 é o fundo e PEN 1 é a cor de primeiro plano quando se liga o Amstrad. O segundo número é a cor que queremos atribuir a esse número PEN/PAPER; este número é obtido da tabela 1, já apresentada. Vejamos um exemplo:

```
INK 1,5
```

Isto redefine PEN 1 com a cor 5 (malva). Vejamos outro caso:

```
INK 1,24
```

Desta vez, PEN 1 é redefinida como cor 24 (amarelo-brilhante).

O melhor modo para demonstrar o uso das cores é o modo 0, que permite o uso simultâneo de 16 cores. A seguir apresentamos um curto programa que mostra cada uma das cores do modo 0:

```
10 ' CORES DO MODO 0
20 '
30 MODE 0
40 FOR T=0 TO 15
50 PEN T
60 PRINT "COR: ";T
70 NEXT
80 PEN 12
```

O leitor poderá verificar que várias das cores cintilam. Consegue-se este efeito acrescentando um terceiro número à instrução INK; a cor

PEN começa a cintilar entre o segundo número e o terceiro (quando estes dois últimos números são traduzidos em código de cor). Portanto, um comando como INK 0,15,6 levaria PEN 0 (o fundo) a cintilar entre as cores 15 (laranja) e 6 (vermelho-brilhante).

A velocidade de cintilação destas cores pode ser alterada do mesmo modo que as cores do comando BORDER, ou seja, através da instrução SPEED INK. O leitor já compreendeu decerto que a cintilação das cores pode ser muito útil para sublinhar passagens de texto; tendo por outro lado observado o funcionamento da instrução INK e dos comandos com ela relacionados, PEN, PAPER e BORDER, por que não tentar acrescentar alguma cor a programas que já introduziu no seu Amstrad?

O seu Amstrad permite igualmente sobrepor caracteres. De início isto pode parecer não ter muito valor prático. No entanto, pode-se sobrepor um caracter de sublinhado a letras já impressas no visor, sublinhando assim uma palavra ou frase importante. O programa que se segue mostra como isto pode ser feito:

```
10 ' TINTAS TRANSPARENTES
20 '
30 CLS
40 MODE 1
50 LOCATE 12,12
60 PRINT ''Exemplo de uso de sublinhados''
70 PRINT : PRINT ''Tinta transparente produzida por CHR$(22)''
80 LOCATE 12,12
90 PRINT CHR$(22)+CHR$(1); ''————————————————
   ————————————————————''
100 PRINT : PRINT ''————————————————————————
    ————————————''
110 PRINT CHR$(22); CHR$(0)
```

A primeira parte da linha 90 e toda a linha 110 realizam a tarefa pretendida. Imprimindo CHR$(22), o computador pode sobrepor caracteres. No entanto, este efeito deve ser ainda preparado pela função CHR$(1), a seguir a CHR$(22). Para «desligar» o efeito, substitua-se CHR$(1) por CHR$(0).

Escreva PRINT CHR$(22);CHR$(1) directamente no teclado e experimente as formas que podem ser criadas colocando dois ou mais caracteres na mesma posição. Depois de ter experimentado um pouco e formado alguns caracteres estranhos mais por acidente do que de propósito, passe à secção seguinte, onde discutiremos a forma de definir exactamente os caracteres que desejamos.

Gráficos definidos pelo utilizador

Os caracteres que podem ser impressos usando a instrução CHR$ podem ser igualmente redefinidos usando os comandos SYMBOL e SYMBOL AFTER. Qualquer forma que se pretenda desenhar no interior dos confins da rede oito por oito pode ser identificada por qualquer dos caracteres 32 a 255. O leitor poderá definir as suas formas para jogos, línguas estrangeiras... ou simplesmente um tipo de letra diferente.

O comando SYMBOL AFTER especifica primeiro a área de caracteres que se pretende redefinir. Se não for usado o comando SYMBOL AFTER, o computador só permitirá a redefinição dos caracteres a partir de 240. Portanto, se quiser redefinir qualquer das letras maiúsculas, terá de usar algo como SYMBOL AFTER 64. Se tiver redefinido um certo número de caracteres e os quiser passar à forma original sem parar o programa, acrescente a linha seguinte ao seu programa:

```
SYMBOL AFTER 255
```

Esta instrução elimina todos os caracteres redefinidos.

Passemos à instrução que de facto redefine os caracteres: SYMBOL. Possui nove parâmetros, sendo o primeiro o caracter que se vai alterar, e correspondendo os outros oito aos valores de cada linha da rede de oito por oito pontos; estes oito valores de linha podem apresentar qualquer valor entre zero e 255. Observe a rede de exemplo que se apresenta em seguida — verificará que possui linhas e colunas numeradas.

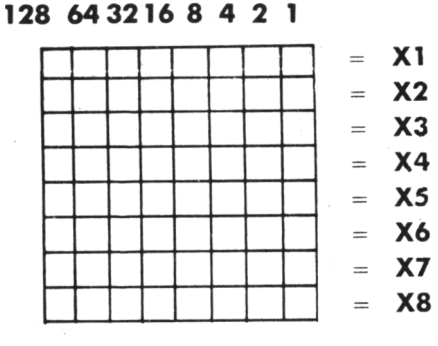


Fig. 1

A instrução SYMBOL terá, portanto, a seguinte aparência: SYMBOL N,X1,X2,X3,X4,X5,X6,X7,X8. Mas quais são os números que definem as colunas?

Observemos primeiro a rede da figura anterior; é formada por 64 quadrados, que podem estar «acesos» (a negro) ou «apagados» (a branco, ou de cor igual ao fundo). Se o quadrado da linha X1 e da coluna oito estiver «aceso», o valor dessa linha será aumentado de oito. Qualquer que seja o número no topo da coluna, se esta está «acesa», o valor da linha onde o quadrado se encontra aumenta do valor correspondente. Se, por exemplo, todas as colunas da linha X1 estivessem «acesas», o valor dessa linha seria 255; se nenhuma o estivesse, o valor seria zero.

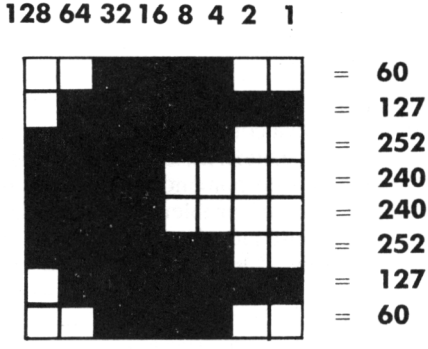


Fig. 2

Ilustremos este aspecto com um exemplo mais adequado. A seguir apresentam-se dois caracteres que podem ser empregues num jogo de acção, uma forma tipo «pacman» e um «invasor do espaço». Ao lado de cada forma encontram-se números correspondentes a cada linha. Se o leitor somar, para cada linha, os valores correspondentes a cada ponto «aceso» (indicados no topo das colunas), obterá o número indicado à direita.

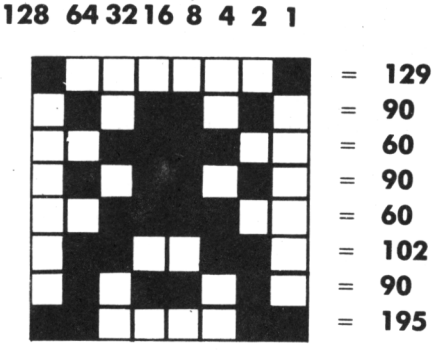


Fig. 3

O leitor já compreende certamente o modo como se criam os gráficos. Uma coisa que o leitor deverá aprender ao criar estes gráficos (UDG's, gráficos definidos pelo utilizador) é que aquilo que desenha num pedaço de papel nem sempre fica com a mesma aparência quando visto no visor. Não se esqueça de que, no Amstrad, apesar de os caracteres serem criados numa rede de oito por oito *pixels*, estes não são de facto quadrados. Por outro lado, há problemas quando se muda de modo gráfico, tendo como consequências que os seus UDG's podem ficar esticados ou achatados. É melhor planear os seus caracteres gráficos numa folha de papel e experimentá-los depois no visor, alterando-os até ficar satisfeito.

Não se esqueça de que pode criar uma forma a partir de vários UDG's usados em conjunto. A seguir apresenta-se um programa que desenha um cavalo e o seu cavaleiro usando 11 UDG's. Por que não, como exercício de programação, acrescentar estes cavalos ao programa de corrida de cavalos apresentado no início do livro?

```
10 ' CAVALO
20 '
30 SYMBOL AFTER 64
40 SYMBOL 65,0,0,0,0,0,0,0,3
50 SYMBOL 66,0,0,0,0,0,1,113,207
```

```
60 SYMBOL 67,0,28,18,42,87,132,15,238
70 SYMBOL 68,0,0,16,248,60,124,199,131
80 SYMBOL 69,5,13,13,9,9,9,9,0
90 SYMBOL 70,9,16,0,144,157,178,192,192
100 SYMBOL 71,221,209,33,1,194,242,11,10
110 SYMBOL 72,128,0,0,0,0,0,0,128
120 SYMBOL 73,3,2,2,2,2,1,0,0
130 SYMBOL 74,192,192,96,32,16,32,0,0
140 SYMBOL 75,13,28,25,19,18,8,0,0
150 CLS
160 LOCATE 18,12
170 PRINT "ABCD"
180 PRINT TAB(18);"EFGH"
190 PRINT TAB(18);"IJK"
```

Para tornar as coisas muito mais simples, vejamos um programa que o ajudará a desenhar os seus caracteres gráficos. Para o usar, deve introduzir um valor para cada linha (entre zero e 255) e o computador converterá o seu número numa versão em larga escala do caracter resultante. Depois de o UDG estar terminado, o computador imprime uma fiada de caracteres no visor, esperando que você carregue na tecla ESC para parar o programa. Os seus UDG's podem assim ser desenhados facilmente...

```
10 REM ++++ CRIADOR DE UDG's ++++
20 DIM G(8),D(8)
30 PRINT "QUE MODO, 1 OU 0" : INPUT L : IF L<>1 AND
   L<>0 THEN PRINT : GOTO 30
40 CLS
50 MODE L
60 PRINT TAB(4);"CRIADOR DE UDG's"
70 PRINT : PRINT : PRINT
80 PRINT "QUE CARACTER QUER"
90 PRINT "DEFINIR"
100 INPUT N : IF N > 255 OR N< 32 THEN GOTO 100
110 SYMBOL AFTER N-1
120 PRINT : PRINT "OK."
130 CLS
140 PRINT TAB(4);"CRIADOR DE UDG's"
150 PRINT : PRINT : PRINT
160 PEN 3
170 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
    CHR$(143);CHR$(143);CHR$(143);CHR$(143);CHR$(143);
    CRH$(143)
```

```
180 FOR T=1 TO 8
190 PRINT CHR$(143);" ";CHR$(143)
200 NEXT
210 PRINT CHR$(143);CHR$(143);CHR$(143);CHR$(143);
    CHR$(143);CHR$(143);CHR$(143);CHR$(143);CHR$(143)
    CHR$(143)
220 PEN 1
230 FOR T=1 TO 8
240 LOCATE 5,20 : PRINT " ":LOCATE 5,20
250 INPUT G(T) : IF G(T) > 255 OR G(T) < 0 THEN 250
260 D(T)=G(T)
270 LOCATE 2,5+T
280 IF D(T) > 127 THEN PRINT CHR$(143); : D(T)=D(T) -128
    ELSE PRINT " ";
290 IF D(T) > 63 THEN PRINT CHR$(143); : D(T) = D(T) -64
    ELSE PRINT " ";
300 IF D(T) > 31 THEN PRINT CHR$(143); : D(T)=D(T) -32
    ELSE PRINT " ";
310 IF D(T) > 15 THEN PRINT CHR$(143); : D(T) = D(T) -16
    ELSE PRINT " ";
320 IF D(T) > 78 THEN PRINT CHR$(143); : D(T) = D(T) -8
    ELSE PRINT " ";
330 IF D(T) > 3 THEN PRINT CHR$(143); : D(T) = D(T) -4
    ELSE PRINT " ";
340 IF D(T) > 1 THEN PRINT CHR$(143); : D(T)=D(T) -2
    ELSE PRINT " ";
350 IF D(T) > 0 THEN PRINT CHR$(143); : D(T) = D(T) -1
    ELSE PRINT " ";
360 NEXT
370 LOCATE 4,24 : PRINT "CARREGUE NUMA TECLA"
380 WHILE INKEY$=" " : WEND
390 SOUND 1,180
400 SYMBOL N,G(1), G(2), G(3), G(4), G(5), G(6), G(7), G(8)
410 CLS
420 PRINT TAB(4);"NÚMERO: ";N
430 PRINT
440 FOR T=1 TO 8 : PRINT "LINHA N. : ";T;"=";G(T) :
    PRINT : NEXT
450 FOR T=1 TO 13 : PRINT CHR$(N); " "; : NEXT
460 ON BREAK GOSUB 480
470 GOTO 460
480 REM +++ RETORNO AO MODO 1 +++
490 MODE 1
500 END
```

Deseja outro tipo de letra?

O leitor poderá aborrecer-se com o tipo de letra empregue pelo Amstrad. Nesse caso, escreva o programa seguinte e redefina o alfabeto, tanto maiúsculo como minúsculo.

O novo tipo de letra é um pouco difícil de descrever — será mais fácil que o leitor escreva o primeiro par de comandos SYMBOL e os execute, para ter uma ideia de qual será o resultado final. Se gostar do que observar no visor, escreva o resto do programa.

```
10 REM REDEFINIR ALFABETO
20 SYMBOL AFTER 64
30 SYMBOL 65,56,100,226,226,254,226,226
40 SYMBOL 66,252,226,226,226,252,226,252
50 SYMBOL 67,124,226,224,224,224,226,124
60 SYMBOL 68,252,226,226,226,226,226,252
70 SYMBOL 69,254,224,224,224,248,224,254
80 SYMBOL 70,254,224,224,224,248,224,224
90 SYMBOL 71,124,226,224,224,238,226,124
100 SYMBOL 72,226,226,226,226,254,226,226
110 SYMBOL 73,56,56,56,56,56,56,56
120 SYMBOL 74,14,14,14,14,142,142,124
130 SYMBOL 75,226,228,232,240,232,228,226
140 SYMBOL 76,224,224,224,224,224,224,254
150 SYMBOL 77,130,198,234,242,226,226,226
160 SYMBOL 78,130,194,226,242,234,230,226
170 SYMBOL 79,124,226,226,226,226,226,124
180 SYMBOL 80,252,226,226,226,252,224,224
190 SYMBOL 81,124,226,226,226,234,228,122
200 SYMBOL 82,252,226,226,226,252,232,228
210 SYMBOL 83,124,226,224,124,14,142,124
220 SYMBOL 84,254,56,56,56,56,56,56
230 SYMBOL 85,226,226,226,226,226,226,124
240 SYMBOL 86,226,226,226,226,100,40,16
250 SYMBOL 87,226,226,226,242,234,198,130
260 SYMBOL 88,134,78,60,56,120,228,194
270 SYMBOL 89,134,78,60,56,56,56,56
280 SYMBOL 90,254,14,28,56,112,224,254
290 SYMBOL 97,0,0,124,14,126,142,126
300 SYMBOL 98,0,224,224,252,226,226,252
310 SYMBOL 99,0,0,124,226,224,226,124
320 SYMBOL 100,0,14,14,126,142,142,126
330 SYMBOL 101,0,0,124,226,252,224,124
340 SYMBOL 102,0,60,114,112,120,112,112
```

```
350 SYMBOL 103,0,0,124,142,142,126,14,124
360 SYMBOL 104,0,224,224,252,226,226,226
370 SYMBOL 105,0,56,0,56,56,56,56
380 SYMBOL 106,0,28,0,28,28,28,156,120
390 SYMBOL 107,0,224,226,228,232,244,226
400 SYMBOL 108,0,112,112,112,112,112,60
410 SYMBOL 109,0,0,244,234,234,234,234
420 SYMBOL 110,0,0,252,226,226,226,226
430 SYMBOL 111,0,0,124,142,142,142,124
440 SYMBOL 112,0,0,252,226,226,252,224,224
450 SYMBOL 113,0,0,126,142,142,126,14,14
460 SYMBOL 114,0,0,252,224,224,224,224
470 SYMBOL 115,0,0,124,224,124,14,124
480 SYMBOL 116,0,112,248,112,112,112,56,0
490 SYMBOL 117,0,0,226,226,226,226,124
500 SYMBOL 118,0,0,226,226,100,56,16
510 SYMBOL 119,0,0,234,234,234,234,116
520 SYMBOL 120,0,0,142,92,56,116,226
530 SYMBOL 121,0,0,142,142,142,126,14,124
540 SYMBOL 122,0,0,254,28,56,112,254
```

Finalmente, vejamos um pequeno programa que acompanha o último e permite definir os números de modo a concordarem com o novo tipo de letra.

```
10 ' DEFINIR ALGARISMOS
20 SYMBOL AFTER 47
30 SYMBOL 48,56,100,226,226,226,100,56
40 SYMBOL 49,12,28,60,92,28,28,28
50 SYMBOL 50,124,142,28,56,112,224,254
60 SYMBOL 51,124,142,14,28,14,142,124
70 SYMBOL 52,14,30,46,78,254,14,14
80 SYMBOL 53,254,224,224,252,14,142,124
90 SYMBOL 54,124,226,224,252,226,226,124
100 SYMBOL 55,254,14,28,56,112,224,224
110 SYMBOL 56,124,142,142,124,142,142,124
120 SYMBOL 57,124,142,142,126,14,142,124
```

Instruções de alta resolução

Como já mencionámos, o Amstrad possui três modos gráficos usando redes de 160 por 200, 320 por 200 e 640 por 200 *pixels*. Sabemos como passar a cada um destes modos e como escolher as cores requeridas,

pelo que podemos descobrir agora quais as instruções que se usam para criar imagens.

Parece apropriado começar pela instrução PLOT. Esta palavra deve ser seguida de dois números — uma coordenada X e uma coordenada Y — e é usada para marcar um *pixel* no visor. A coordenada X é o valor da posição do *pixel* na horizontal, enquanto a coordenada Y é a posição do *pixel* na vertical. Como seria de esperar, o menor valor de X corresponde ao lado esquerdo do visor, começando pelo valor um e terminando em 640.

A coordenada Y funciona de um modo um pouco diferente do que seria de esperar; o seu valor menor corresponde à parte inferior do visor (valor um), e aumenta até atingir o topo (400). Parece haver aqui uma anomalia: se a resolução vertical do Amstrad é 200, como podem as coordenadas verticais variar entre 1 e 400? Bom, a resposta simples é que o melhor é perguntar a quem escreveu a linguagem BASIC Locomotive! Do ponto de vista do programador, no entanto, existe de facto uma resolução de 200 *pixels*, sendo cada *pixel* representado por dois números. Por exemplo, se escrevermos PLOT 1,2 e PLOT 1,3, não notaremos qualquer diferença no visor dado que ambos os números se aplicam ao mesmo *pixel*. No entanto, note-se que isto só acontece no caso das coordenadas verticais.

A seguir às duas coordenadas da instrução PLOT pode-se usar opcionalmente um parâmetro que determina a cor do *pixel*. Escreva a rotina seguinte:

```
10 ' DEMONSTRAÇÃO DA COR
20 '
30 MODE 0
40 INK 0,0
50 FOR T=1 TO 15
60 PLOT 320,200,T : PLOT 324,200,T
70 FOR D=1 TO 1000 : NEXT
80 NEXT T
```

A instrução PLOT pode ser usada em diversas situações. Desloca a posição do cursor gráfico e pode ser usada para o colocar em qualquer posição do visor. PLOT pode também ser usada para criação de formas muito complexas. Esta última função é ilustrada na rotina que se segue; esta desenha um cone tridimensional no quadrante inferior esquerdo do visor. Os cálculos que o Amstrad deve executar para criar este tipo de imagem são bastante complexos, pelo que o leitor deverá esperar um pouco para poder observar o resultado. E considere-se com sorte — quando um programa deste tipo foi posto em execução num ORIC 1, demorou 10 minutos a terminar!

```
10 ' CONE 3D
20 '
30 DEFINT A-Z
40 CLS
50 MODE 1
60 FOR A=-100 TO 100
70 J=0 : K=1 : T=10
80 V=T*INT(SQR(10000-A*A)/T)
90 FOR B=V TO -V STEP -T
100 C=INT(80+30*SIN((SQR(A*A+B*B))/12)-0.7*B)
110 IF C< J THEN 150
120 J=C
130 PLOT A+110,C-15,1
140 K=0
150 NEXT : NEXT
160 SOUND 1,100,300,15
```

Pode-se usar igualmente o comando PLOT como base de uma curta rotina que desenha círculos. A BASIC do Amstrad não possui uma instrução para construção directa de círculos, mas a rotina que se segue é bastante boa para esse efeito e para a criação de elipses:

```
10 ' ROTINA DE DESENHO DE CÍRCULOS
20 '
30 CLS
40 FOR C=1 TO 360
50 DEG
60 PLOT 320,200,0
70 PLOT 320+100*COS(C),200+100 * SIN(C),1
80 NEXT
```

A linha mais importante do programa anterior é a 70, onde a posição de «desenho» é calculada e o *pixel* marcado. Vejamos melhor. O primeiro número, 320, indica a posição horizontal do centro do círculo; no caso da rotina acima, escolheu-se o centro do visor. O número seguinte, 100, é o raio do círculo. A segunda parte da linha mostra a posição vertical do centro do círculo, e de novo o seu raio.

Tendo esta informação em mente, é muito fácil desenhar círculos à nossa vontade. Altere o 320 e o 200 da linha 70, e passe os valores da linha 60 para as coordenadas do centro do seu círculo. Modifique agora as dimensões do círculo alterando os dois valores 100 da linha 70. Poderá também desenhar ovais dando simplesmente aos valores do raio na linha 70 grandezas diferentes; experimente alterar apenas o primeiro valor para 70 e veja o que acontece.

Se o primeiro número for inferior ao segundo, deverá obter uma forma oval na vertical; se o menor valor for o segundo, a oval ficará disposta na horizontal.

O comando PLOTR é um familiar próximo de PLOT. A diferença é que PLOTR marca um ponto numa posição *relativa* ao cursor gráfico, enquanto PLOT utiliza a posição absoluta do cursor. Vejamos o que se passa com um exemplo.

A posição do cursor é (320,200). Assim, PLOT 10,10 marcaria um *pixel* no canto inferior esquerdo do visor, passando o cursor para (10,10). PLOTR, por outro lado, marcaria um ponto 10 *pixels* acima e 10 *pixels* à direita da posição actual do cursor. Esta passaria a ser (330,210).

PLOTR pode ser usada na rotina de desenho do círculo. De facto, encurta-a ligeiramente, como se pode ver:

```
10 ' ROTINA DE DESENHO DE CÍRCULO USANDO PLOTR
20 '
30 CLS
40 FOR C=1 TO 360
50 DEG
60 PLOT 320,200,0
70 PLOTR 100*COS(C),100*SIN(C),1
80 NEXT
```

Desenho e teste

Passamos a três outras instruções, DRAW, MOVE e TEST. Todas tratam a posição absoluta do cursor e todas dispõem de uma versão que utiliza a posição relativa do cursor. Se quisermos activar as versões relativas das instruções, basta-nos acrescentar um «R» ao nome de cada uma delas.

A instrução DRAW permite-nos construir uma linha desde a posição do cursor até qualquer ponto especificado no visor. DRAW é uma instrução muito útil, permitindo obter vários efeitos interessantes. Observaremos alguns deles dentro em pouco, mas, por agora, convirá esclarecer a sintaxe deste comando:

```
DRAW X,Y,C
```

onde X é a coordenada X da nova posição do cursor, Y é a coordenada Y do novo cursor e C é um parâmetro opcional que indica a cor da linha. A linha é desenhada entre a antiga posição do cursor e a nova posição deste. Se optarmos pelo uso de DRAWR, tal como no caso de PLOTR, os valores de X e Y são somados à antiga posição de cursor. Observemos isto no programa seguinte:

```
10 ' A CAVERNA LUMINOSA
20 '
30 Z=100 : F=0
40 CLS
50 GOSUB 120
60 WHILE INKEY$=" "
70 F=F+1 : IF F=20 THEN F=0 : GOSUB 120
80 PLOT 320,200
90 DRAW INT(RND*640)+1,INT(RND*200)+1,INT(RND*3)+1
100 WEND
110 END
120 FOR T=1 TO 180 STEP 2
130 DEG
140 PLOT 320,200
150 PLOTR Z*COS(T),Z*SIN(T),1
160 NEXT
170 Z=Z-INT(Z/15)
180 IF Z<25 THEN 200
190 RETURN
200 SOUND 1,350,100
210 GOTO 210
```

O leitor necessitará de um *joystick* para o programa seguinte, se bem que este possa ser facilmente convertido para uso com teclado, recorrendo a INKEY$. Basta movimentar o cursor pelo visor e tocar no botão de «tiro» quando se pretende desenhar uma linha.

```
10 ' DESENHAR FIGURA
20 '
30 CLS
40 X=32*0 : Y=200
50 N=JOY(0)
60 IF N=1 AND Y< 400 THEN Y=Y+1
70 IF N=2 AND Y> 1 THEN Y=Y-1
80 IF N=4 AND X> 1 THEN X=X-1
90 IF N=8 AND X< 640 THEN X=X+1
100 IF N=16 THEN DRAW X,Y
110 GOTO 50
```

O comando MOVE coloca simplesmente o cursor gráfico numa nova posição sem actuar sobre o *pixel*. O comando MOVER é a versão relativa de MOVE.

TEST determina a cor de um dado ponto do visor. A sintaxe de TEST e TESTR é ligeiramente diferente de MOVE e DRAW, e como é evi-

dente nenhum dos comandos de teste de cor altera a posição do cursor. Por exemplo, PRINT TEST (X,Y) dará a cor do *pixel* com as coordenadas (X,Y).

Vejamos um programa que mostra a instrução TEST em funcionamento:

```
10 ' CORES
20 '
30 A=10
40 B=200
50 CLS
60 FOR T=580 TO 640
70 PLOT T,1,3
80 DRAW T, 400, 3
90 NEXT
100 PLOT 1,1,1
110 TAG
120 WHILE TEST (A+14,B<>3)
130 PLOT A,B
140 PRINT ">"
150 A=A+2
160 WEND
170 TAGOFF
180 LOCATE 15,12
190 PRINT "CRRAAASSSHHHH!!"
```

TAG e TAGOFF, XPOS e YPOS

Como o leitor provavelmente notou no último programa, existe uma instrução designada TAG. Que faz ela? Liga um dado *stream* a um cursor gráfico, de tal modo que é possível imprimir caracteres na posição do cursor. No programa anterior pode observar-se um uso imediato desta instrução — permite deslocar caracteres no visor com muita flexibilidade e depois, usando a instrução TEST, verificar se o caracter está prestes a entrar em contacto com uma cor diferente; TAG é extremamente valioso quando se escrevem programas de jogo.

O comando TAG deve ser seguido de um número de *stream*; se nenhum for especificado, o computador considera que se trata de *stream* zero (correspondente ao visor de texto normal). TAGOFF pode também ser observado no programa anterior. Faz exactamente o oposto de TAG, destruindo qualquer ligação entre o texto e o cursor gráfico.

Uma outra característica dos grafismos Amstrad é a possibilidade de verificar a posição do cursor gráfico. Como o cursor usado para esta tarefa é invisível, é necessário dispor de dois comandos que nos possam indicar sempre onde se encontra.

XPOS indica o valor horizontal do cursor invisível, enquanto YPOS trata o valor vertical. Estes podem ser usados para criar um efeito de «rotação» do visor gráfico. No visor de texto, quaisquer caracteres que tentem continuar para além do rebordo direito do visor são transferidos para a linha imediatamente abaixo. Usando TAG, juntamente com XPOS e YPOS, podemos criar o mesmo efeito no visor gráfico.

```
10 ' TEXTO NO VISOR GRÁFICO
20 '
30 A=10 : B=200
40 CLS
50 TAG
60 RESTORE 180
70 A=10
80 FOR T=1 TO 47
90 IF XPOS > 620 THEN A=10 : B=B-40
100 MOVE A,B
110 IF YPOS < 41 THEN B=360 : GOTO 100
120 READ H$
130 PRINT H$;
140 A=A+40
150 NEXT
160 WHILE INKEY$= " " : WEND
170 GOTO 60
180 DATA E,S,T,E," ",P,R,O,G,R,A,M,A," "," "," ",E,S,P,A,
    Ç,A," ",T,E,X,T,O," ",N,O," "," "," ",V,I,S,O,R," ",
    G,R,Á,F,I,C,O
```

Vejamos ainda um par de rotinas gráficas que o divertirão. Introduza-as no computador e experimente-as.

```
10 ' LEONARDO
20 '
30 MODE 0
40 CLS
50 A=INT(RND*640)+1
60 B=INT(RND*400)+1
70 C=INT(RND*15)+1
80 IF RND < 0.4 THEN PLOT A,B,C : GOTO 110
90 IF RND < 0.8 AND RND > 0.39 THEN DRAW A,B,C :
   GOTO 110
100 IF RND > O.79 THEN GOSUB 140 : GOTO 110
110 SOUND 1,INT(RND*600)+60,5
120 GOTO 50
```

```
130 '
140 ' DESENHAR CURVA
150 '
160 FOR T=1 TO INT(RND*315)+45 STEP 2
170 DEG
180 PLOT A,B,C
190 PLOTR 100*COS(T),90*SIN(T),C
200 NEXT
210 RETURN
```

O último programa produziu algumas imagens interessantes — experimente o seguinte para obter formas mais ordenadas. Basta carregar numa qualquer tecla para obter a imagem seguinte:

```
10 ' FIGURAS DE 3 A 12 LADOS
20 '
30 FOR T=3 TO 12
40 MODE 1
50 ORIGIN 320,200
60 R=100
70 GOSUB 150
80 WHILE INKEY$=" " : WEND
90 ORIGIN 0,0
100 NEXT
110 END
120 '
130 ' DESENHAR FORMA
140 '
150 DEG
160 MOVER 0,R
170 FOR N=0 TO 360 STEP INT (360/T)
180 DRAW R*SIN(N), R*COS(N)
190 NEXT
200 LOCATE 10,24
210 PRINT "UMA FIGURA DE ";T;" LADOS"
220 RETURN
```

A janela gráfica

Tal como as outras janelas disponíveis no Amstrad, esta deve ser definida em termos de dimensões. Se se usa a janela gráfica, todos os gráficos são impressos nessa janela, não podendo ocorrer em qualquer ponto do visor.

ORIGIN é uma instrução que constrói a janela gráfica, sendo seguida de seis números, como se mostra em seguida:

ORIGIN coordenada X, coordenada Y, esquerda, direita, topo, fundo

Os últimos quatro valores determinam as dimensões e posições da janela gráfica e, depois de estas terem sido definidas, o computador pode alterá-las ligeiramente de modo a facilitar as construções na janela gráfica. Uma aplicação útil disto é no campo da estatística, onde se podem apresentar os dados numéricos numa porção do visor, enquanto noutra se mostra uma representação gráfica desses mesmos dados.

O comando CLG assemelha-se a CLS, mas é usado para limpar apenas a janela gráfica.

Vejamos um curto programa que usa ORIGIN e CLG para criar uma janela dentro da qual é apresentado um gráfico:

```
10 ' A JANELA GRÁFICA
20 '
30 INK 0,0
40 CLS
50 PEN 3
60 LOCATE 1,1
70 FOR T=1 TO 25 : PRINT "TEXTO:TEXTO:TEXTO:TEXTO:
   TEXTO:TEXTO:TEXTO:TEXTO:";
80 NEXT
90 ORIGIN 0,0,10,630,200,10
100 CLG
110 FOR T=30 TO 600
120 R=104+90*SIN(2*PI*T/600)
130 PLOT T,R,2
140 NEXT
150 GOTO 150
```

Animação

Já observámos vários exemplos de gráficos em movimento neste livro, desde o simples efeito de centragem em «Cavalos» até ao movimento mais complicado de alguns jogos de acção. Por outro lado, já vimos como é possível realizar movimento com a instrução LOCATE, e como se pode usar TAG para dar a humildes caracteres o movimento geralmente associado ao do visor gráfico.

O que ainda não vimos foi exemplos de animação em tempo real. A animação é uma área difícil dos computadores domésticos — podem-se conseguir bons resultados, mas, geralmente, são necessárias muitas horas de trabalho para se obter êxito. Devido à necessidade de realizar movimentos extremamente rápidos, verificamos normalmente que só o código-máquina permite obter resultados satisfatórios. No

entanto, existem certos tipos de animação computadorizada em que o nosso Amstrad pode dar efeitos excelentes.

Recordará o leitor o cavalo que criámos usando alguns UDG's? Bom, essa figura vai fazer um regresso triunfal no próximo programa. A técnica usada neste programa é a mais simples forma de animação disponível em computadores — o Amstrad desenha uma forma; pausa durante algum tempo; altera a posição da forma e desenha-a de novo, apagando a anterior. O principal acrescento ao programa é o movimento tanto das pernas do cavalo e do cavaleiro como do conjunto da figura no visor.

```
10 ' CAVALO EM MOVIMENTO
20 '
30 DEFINT T
40 ON BREAK GOSUB 430
50 SYMBOL AFTER 64
60 SYMBOL 65,0,0,0,0,0,0,0,3
70 SYMBOL 66,0,0,0,0,0,1,113,207
80 SYMBOL 67,0,28,18,42,87,132,15,238
90 SYMBOL 68,0,0,16,248,60,124,199,131
100 SYMBOL 69,5,13,13,9,9,9,9,0
110 SYMBOL 70,9,16,0,144,157,178,192,192
120 SYMBOL 71,221,209,33,1,194,242,11,10
130 SYMBOL 72,128,0,0,0,0,0,0,128
140 SYMBOL 73,3,2,2,2,2,1,0,0
150 SYMBOL 74,192,192,96,32,16,32,0,0
160 SYMBOL 75,13,28,25,19,18,8,0,0
170 SYMBOL 76,128,128,128,0,0,0,0,0
180 SYMBOL 77,16,0,16,144,177,167,64,128
190 SYMBOL 78,69,1,1,1,194,246,20,20
200 SYMBOL 79,128,0,0,0,0,0,0,0
210 SYMBOL 80,128,128,128,128,128,128,64,0,0
220 SYMBOL 81,28,28,24,24,20,12,10,0
230 INK 0,9
240 INK 1,0
250 CLS
260 PLOT 1,300 : DRAW 640,300,3
270 PLOT 1,100 : DRAW 640,100,3
280 FOR T=1 TO 34
290 LOCATE T,12
300 PRINT "ABCD"
310 LOCATE T,13
320 PRINT "EFGH"
330 LOCATE T,14
```

```
340 PRINT "IJKL"
350 FOR H=1 TO 60 : NEXT
360 LOCATE T,13
370 LOCATE T,13
380 PRINT "EMNO"
390 LOCATE T,14
400 PRINT "IPQ"
410 FOR H=1 TO 60 : NEXT
420 NEXT
430 SYMBOL AFTER 255
```

Pode-se obter uma forma de animação mais complexa usando as cores INK. Os dois programas que se seguem demonstram este método admiravelmente, desenhando um cilindro cujos segmentos coloridos se deslocam suavemente.

Ambos os programas consistem essencialmente em duas partes principais. A primeira parte desenha o cilindro e «pinta» os segmentos (quatro segmentos no primeiro programa e oito no segundo), enquanto a outra realiza a animação. É a segunda parte do programa que por agora mais nos interessa. Escreva o primeiro programa e volte ao texto para compreender o que se está a passar.

```
10 ' CILINDRO EM MOVIMENTO
20 '
30 GOSUB 240
40 GOSUB 60
50 '
60 ' REALIZAR A ANIMAÇÃO
70 '
80 D=1 : H=1
90 FOR T=1 TO 4 : READ C(T) : NEXT
100 DATA 24,2,6,22
110 WHILE INKEY$=" "
120 FOR F=1 TO 4
130 D=D+1 : IF D=5 THEN D=1
140 INK F,C(D)
150 NEXT
160 ENT 1,5,-1,51 : ENV 1,3,-2,85
170 SOUND 1,150,5,11,1,1,5
180 FOR G=1 TO 300 : NEXT
190 H=H+1 : IF H=5 THEN H=1
200 D=H
210 GOTO 120
220 WEND
```

```
230 '
240 ' DESENHAR
250 '
260 MODE 0
270 INK 1,24 : INK 2,2 : INK 3,6 : INK 4,22
280 CLS
290 C=1
300 FOR T=1 TO 360
310 PLOT 320,100,0
320 DEG
330 IF T>90 THEN C=2 : IF T>180 THEN C=3 : IF T>270
    THEN C=4
340 PLOT 320+100*COS(T),100+100*SIN(T),C
350 DRAWR 0,150,C
360 NEXT
370 RETURN
```

As cores requeridas são lidas para as instruções INK. Depois as INK são alteradas num ciclo; portanto, o quadrante que era INK 3 passa a INK 4, o INK 4 passa a INK 1, e assim por diante. Nestas condições ocorre um deslocamento das cores e, se se escolhe uma pausa apropriada, parece que o cilindro está a rodar.

O segundo dos programas produz um efeito melhor por usar oito segmentos coloridos. Estas cores de INK são novamente lidas para os primeiros oito valores e, usando o mesmo princípio de anteriormente, um ciclo acrescenta um a cada valor, de tal modo que esta se desloca de uma posição em torno do cilindro, na direcção contrária ao ponteiro do relógio.

Experimente com estes programas alterando as formas e cores e depois, quando tiver compreendido bem o princípio de animação tratado, poderá criar efeitos próprios.

```
10 ' CILINDRO EM MOVIMENTO 2
20 '
30 GOSUB 220
40 GOSUB 60
50 '
60 ' REALIZAR ANIMAÇÃO
70 '
80 D=1 : H=1
90 FOR T=1 TO 8 : READ C(T) : NEXT
100 DATA 24,2,6,9,15,22,13,7
110 ON BREAK GOSUB 390
120 FOR F=1 TO 8
```

```
130 D=D+1 : IF D=9 THEN D=1
140 INK F,C(D)
150 NEXT
160 FOR T=1 TO 30 : NEXT
170 H=H+1 : IF H=9 THEN H=1
180 D=H
190 GOTO 120
200 END
210 '
220 ' DESENHAR
230 '
240 MODE 0
250 INK 1,24 : INK 2,2 : INK 3,6 : INK 4,9
260 INK 5,15 : INK 6,22 : INK 7,13 : INK 8,7
270 CLS
280 C=1
290 FOR T=1 TO 360
300 PLOT 320,20,0
310 DEG
320 IF T > 45 THEN C=2 : IF T > 90 THEN C=3 : IF T > 135
    THEN C=4
330 IF C=4 AND T > 180 THEN C=5 : IF T > 225 THEN C=6
    : IF T > 270 THEN C=7 : IF T > 315 THEN C=8
340 MOVE 320+200*COS(T),150+100*SIN(T)
350 DRAWR 0,150,C
360 NEXT
370 RETURN
380 '
390 ' RETORNO AO NORMAL
400 '
410 MODE 1
420 INK 1,24 : INK 3,6
430 END
```

Em conclusão, este capítulo deve servir como instrução à criação dos efeitos gráficos do próprio leitor. Agora que conhece comandos como DRAW, PLOT e TEST, deve ser capaz de construir as suas próprias imagens. Um bom ponto de partida consiste em desenhar uma imagem de um cavalo usando apenas linhas rectas. Pode depois passar a figuras mais complexas, talvez incorporadas num outro programa, por exemplo uma aventura de algum tipo.

O último programa desta secção poderia encontrar-se no capítulo sobre código-máquina. Trata-se de uma rotina em código que lhe permite guardar uma imagem inteira na memória do computador e chamá-la

instantaneamente ao visor. Várias aplicações se tornam imediatamente óbvias, mas poderia ser usada, por exemplo, num programa de trabalho artístico, onde o utilizador pudesse desenhar uma imagem, guardá-la em fita, carregá-la para a memória e passá-la instantaneamente para o visor. Os jogos que exigem imagens complicadas podem ser carregados enquanto o visor apresenta as instruções de jogo e depois, repentinamente, quando a imagem é precisa, aparece instantaneamente.

Escreva o programa que se segue e execute-o. Quando tiver terminado a imagem pretendida, escreva CALL 27488 para guardar a imagem; quando quiser recuperá-la, basta escrever CALL 27500, o que a trará imediatamente ao visor.

Este programa é muito pequeno, podendo, por exemplo, ser renumerado a partir da linha 20 000 e usado em qualquer dos seus programas.

```
10 MEMORY 27487
20 FOR T=27488 TO 27511
30 READ X : POKE T,X : NEXT
40 DATA 1,0,64,33,0,192,17,122,107,237,176,201
50 DATA 1,0,64,33,122,107,17,0,192,237,176,201
```

16

SONS

As capacidades sonoras do Amstrad poderiam encher um livro por si sós, tal é a sua complexidade e potencial. Vejamos, no entanto, o que é possível tratar aqui.

Primeiro observemos a instrução que nos permite produzir sons isolados e melodias simples. É designada SOUND e, no caso de um tom simples, é seguida de dois parâmetros:

```
SOUND CH,P
```

onde CH é o canal escolhido e P o tom especificado. Comecemos por tratar os canais. O Amstrad possui quatro canais — três canais sonoros e um canal de ruído que pode ser misturado com o som. No entanto, por agora, consideremos apenas os canais sonoros. Como o leitor talvez já saiba, o seu Amstrad permite o uso da estereofonia, o que pode ser verificado ligando um gravador estereofónico (ou um sistema de alta fidelidade) à tomada apropriada na parte traseira do Amstrad.

Dispomos de sete canais com versões estereofónicas e mono: os canais um, dois e quatro são mono, enquanto os canais três, cinco e seis misturam dois canais; o canal sete junta os três canais, produzindo uma espécie de som estereofónico.

A tonalidade sonora pode variar entre zero e 4095. No entanto, os valores extremos são pouco úteis; geralmente serão suficientes valores entre 40 e 900. Os sons de demonstração que se seguem usam a forma mais simples de instrução SOUND:

```
10 SOUND 1,100
20 WHILE INKEY$=" " : WEND
30 SOUND 1,500
40 WHILE INKEY$=" " : WEND
50 SOUND 1,300
60 END
```

O Amstrad começa por tocar um curto *beep* de tom agudo, esperando que o utilizador carregue numa tecla. Depois de isto acontecer, é tocada uma nota muito baixa e, se se carrega numa nova tecla, é tocada uma nota intermédia.

Da demonstração anterior conclui-se que os maiores valores do parâmetro produzem tons mais baixos. Mas a instrução SOUND pode ser igualmente usada no interior de um ciclo para criar um crescendo ou diminuendo, como poderá verificar usando os programas que se seguem:

```
10 FOR T=60 TO 250
20 SOUND 1,T
30 NEXT
```

```
10 FOR T=250 TO 60 STEP -1
20 SOUND 1,T,2
30 NEXT
```

O segundo destes programas possui o número 2 após a instrução SOUND 1,T. Este novo parâmetro indica a duração da nota em centésimos de segundo. Se bem que o valor da linha 20 produza uma nota muito curta, quando esse valor é usado num ciclo contínuo, em que a tonalidade seja aumentada ou diminuída em cada passagem, obtemos um som contínuo cada vez mais agudo ou grave. Se tornarmos o ciclo da linha 10 bastante menor, por exemplo FOR T=120 TO 70, e encurtarmos o tom do som para 1, obteremos um som de tipo *zap*, que podemos usar para representarmos um *laser* num jogo.

O leitor pode evidentemente não desejar um som contínuo, e sim uma série de notas distintas. Se for este o caso, experimente usar a instrução STEP num ciclo FOR/NEXT. A dimensão STEP indica a diferença entre as notas, como se mostra na curta rotina seguinte:

```
10 FOR T=60 TO 400 STEP 20
20 SOUND 1,T
30 NEXT
```

Note que o valor da duração na rotina anterior foi omitido; se não é especificado qualquer valor, o computador escolhe um quinto de segundo.

Estas notas isoladas, no entanto, nada têm a ver com música. Mas antes de podermos construir uma melodia necessitamos de conhecer os valores correspondentes e cada nota musical; felizmente, a *Amsoft* (departamento de *software* da Amstrad) indica esses valores nas costas do manual Amstrad. Vejamos apenas um par de notas que ilustram este aspecto; nesta fase, poderá ser uma boa ideia pegar no manual e consultar as indicações nele dadas para os diversos tons.

Dó principal = 478
Lá = 284

A partir destas tabelas, torna-se muito simples construir uma subrotina com notas de várias oitavas que possa ser usada para tocar música. Vejamos um exemplo:

```
10 ' MÚSICA
20 '
30 GOSUB 1000
100 ' TOCAR NOTAS
110 FOR T=1 TO 24
120 SOUND 1,M(T)
130 NEXT
990 END
1000 '
1010 ' GUARDAR NOTAS
1020 '
1030 DIM M(24)
1040 FOR T=1 TO 24
1050 READ M(T)
1060 NEXT
1070 DATA 478,451,426,402,379,358,338,319,301,284,268,253
1080 DATA 239,225,213,201,190,179,169,159,150,142,134,127
1090 RETURN
```

Os dados (os valores das notas) são lidos para um *array* M(n), e em seguida a rotina limita-se a tocar cada nota por ordem ascendente, um pouco como uma escala. Estas notas podem ser tocadas pela instrução SOUND escolhendo a nota apropriada e substituindo o valor requerido; assim, se quisermos tocar o menor dó permitido na rotina, tocamos a primeira nota SOUND 1,M(1) seguida da duração desejada.

Tocar uma melodia usando este método envolve o uso de várias instruções DATA, cada uma delas contendo o elemento musical correcto. É evidente que este não é o método mais eficiente em termos de uso da memória da máquina — apresenta-se em seguida um outro, bastante melhor.

Neste caso, o computador lê um segundo conjunto de dados para o *array*, correspondentes às notas do *array* musical. Portanto, se as primeiras três notas que quisermos tocar forem a primeira, a sétima e a terceira do *array* de notas, o *array* da melodia deve possuir como primeiros elementos os valores 1, 7 e 3.

Escreva o programa que se segue, execute-o e ouvirá uma composição conhecida.

```
10 ' SUBROTINA MUSICAL
20 '
30 GOSUB 240
40 ' TOCAR NOTAS
50 DIM P(100)
60 RESTORE 100
70 FOR T=1 TO 47
80 READ P(T)
90 NEXT
100 DATA 1,1,8,8,10,10,8,0,6,6,5,5,3,3,1
110 DATA 0,8,8,6,6,5,5,3,0,8,8,6,6,5,5,3
120 DATA 0,1,1,8,8,10,10,8,0,6,6,5,5,3,3,1
130 '
140 ' 0 = PAUSA
150 '
160 FOR T=1 TO 47
170 SOUND 1,M(P(T))
180 SOUND 1,0,10
190 '
200 ' SOUND 1,0 PRODUZ UM INTERVALO
210 '
220 NEXT
230 END
240 '
250 ' GUARDAR
260 '
270 DIM M(24)
280 RESTORE 320
290 FOR T=1 TO 24
300 READ M(T)
310 NEXT
320 DATA 478,451,426,402,379,358,338,319,301,284,268,253
330 DATA 239,225,213,201,190,179,169,159,150,142,134,127
340 RETURN
```

Substitua agora a rotina de tocar pela dada em seguida, e ouvirá uma nova melodia. Não se esqueça de que deve renumerar a rotina que guarda as notas, de tal modo que se inicie na linha 290. Isto pode ser feito escrevendo RENUM 290,240 antes de ser escrita a nova rotina.

```
10 ' SUBROTINA MUSICAL
20 '
30 GOSUB 290
40 ' TOCAR NOTAS
```

```
50 DIM P(120)
60 RESTORE 100
70 FOR T=1 TO 105
80 READ P(T)
90 NEXT
100 DATA 8,9,10,18,0,10,18,0,10,18,0,0
110 DATA 18,20,21,22,18,20,22,0,17,20,0,18,0,0
120 DATA 8,9,10,18,0,10,18,0,10,18,0,0
130 DATA 15,13,12,15,18,22,0,20,17,15,20,0,0
140 DATA 8,9,10,18,0,10,18,0,10,18,0,0
150 DATA 18,20,21,22,18,20,22,0,17,20,0,18,0,0
160 DATA 18,20,21,22,18,20,22,0,18,20,0,22,18,20,22,0,18,20
170 DATA 0,22,18,20,22,0,17,20,0,18
180 '
190 ' 0=PAUSA
200 '
210 FOR T=1 TO 105
220 SOUND 1,M(P(T))
230 SOUND 1,0,4
240 '
250 ' SOUND 1,0 CRIA UM INTERVALO
260 '
270 NEXT
280 END
```

A terceira e última melodia é estritamente para ouvir nos meses de Inverno (estou certo de que o leitor a reconhecerá). Uma vez mais será necessário renumerar a rotina que guarda os sons — elimine, portanto, a rotina que toca a melodia escrevendo DELETE 10-280 e depois execute a operação de renumeração escrevendo o comando RENUM 230,290.

```
10 ' SUBROTINA MUSICAL
20 '
30 GOSUB 230
40 ' TOCAR NOTAS
50 DIM P(120)
60 RESTORE 100
70 FOR T=1 TO 28
80 READ P(T)
90 NEXT
100 DATA 10,10,10,0,10,10,10,0,10,13,6,8,10
110 DATA 0,0,11,11,11,11,11,10,10,10,13,13,11,8,6
120 '
```

```
130 ' 0=PAUSA
140 '
150 FOR T=1 TO 28
160 SOUND 1,M (P(T))
170 SOUND 1,1,8,0
180 '
190 ' SOUND 1,0 CRIA UM INTERVALO
200 '
210 NEXT
220 END
```

Volume

O parâmetro de volume do som é o quarto valor da instrução SOUND. Se bem que exista um controlo de volume no Amstrad (que se encontra do lado direito da caixa, junto ao interruptor que liga a máquina), pode-se controlar o volume de som usando *software*. O parâmetro de volume pode variar entre 1 e 15, sendo este último o mais elevado; o computador selecciona automaticamente um valor médio e, de facto, este é um dos parâmetros menos usado da instrução SOUND. No entanto, é muito melhor usar um valor médio para a maioria dos sons no interior de um programa — desse modo há sempre «folga» disponível para produzir um som alto, surpreendente (por exemplo, uma explosão), ou um som baixo. Em seguida apresentamos um efeito interessante, conseguido usando o nível de volume:

```
10 ' DESAPARECIMENTO DO SOM
20 '
30 FOR T=7 TO 1 STEP -1
40 SOUND 1,200-CINT(3.5*T),20,T
50 NEXT
```

Componha as suas músicas

O programa que se segue permite ao leitor construir as suas próprias composições, que depois serão tocadas pelo próprio computador. As notas são escritas sob a forma de letras, usando-se um espaço para cada pausa. Uma melodia simples poderá ser escrita AB DCEFDC CDFEGABCC. Pode-se introduzir até 40 notas, se bem que este número possa ser facilmente aumentado usando o modo 2 ou construindo duas pautas na imagem, dado que as notas são representadas sobre pauta enquanto são tocadas.

Depois de o computador ter aceite a sua música (recusando quaisquer letras incorrectas), ser-lhe-á pedido que indique a velocidade de execução da música e o volume a que deseja que a sua obra-prima seja tocada.

Se o leitor não ficar satisfeito com o resultado quando as notas forem tocadas, pode alterar a duração das pausas ou observar a secção sobre efeitos sonoros que se encontra mais adiante, neste capítulo, transformando o seu Amstrad num sintetizador. Este programa tem bastantes possibilidades, e tudo depende de si!

```
10 ' COMPOR MÚSICA
20 '
30 MODE 1
40 WINDOW # 1,3,37,22,25
50 INK 3,24
60 PAPER # 1,3
70 PEN # 1,1
80 INK 1,0 : INK 0,26
90 Y=0
100 GOSUB 980
110 CLS
120 PRINT TAB(10); ''COMPOSITOR''
130 PRINT TAB (10);''==========''
140 PRINT : PRINT : PRINT
150 CLS#1
160 PRINT # 1, ''Dó a Si inferiores, escreva C a B''
170 PRINT # 1, ''Dó superior, escreva H. Ré superior,''
180 PRINT # 1, ''escreva I. Mi superior, escreva J. Fá superior,
    escreva K''
190 INPUT ''Música'';A$
200 A$=UPPER$(A$)
210 INPUT ''TEMPO (ENTRE 1 e 40)'';P
220 IF P > 40 OR P < 1 THEN PRINT : GOTO 210
230 INPUT ''VOLUME (ENTRE 1 e 15)'';V
240 IF V > 15 OR V < 1 THEN PRINT : GOTO 230
250 GOSUB 790
260 X=1
270 IF LEN(A$) > 38 THEN A$=LEFT$(A$,38)
280 IF LEN(A$) > 19 THEN F=1 ELSE F=2
290 FOR T=1 TO LEN(A$)
300 IF ASC(MID$(A$,T,1))=32 THEN FOR J=1 TO 10*P : NEXT
    : SOUND 1,0,P*2,0 : GOTO 370
310 IF ASC(MID$(A$,T,1)) < 64 OR ASC(MID$(A$,T,1)) > 75
    THEN 390
```

```
320 ON ASC(MID$(A$,T,1))-64 GOSUB 450,480,510,540,570,
    600,660,690,720,750
330 LOCATE X,Y : PRINT CHR$(237);
340 Z=Y-6 : Z=13-Z
350 FOR J=1 TO 10*P : NEXT
360 SOUND 1,M(Z),P*2,V
370 FOR J=1 TO 10*P : NEXT
380 X=X+F
390 NEXT
400 CLS # 1
410 INPUT # 1, "DESEJA OUVIR DE NOVO (R) OU RECOME-
    ÇAR (S) ";J$
420 IF J$="S" THEN RUN
430 IF J$="R" THEN GOSUB 790 : GOTO 260
440 GOTO 400
450 ' Lá
460 Y=12
470 RETURN
480 ' Si
490 Y=11
500 RETURN
510 ' Dó
520 Y=17
530 RETURN
540 ' Ré
550 Y=16
560 RETURN
570 ' Mi
580 Y=15
590 RETURN
600 ' Fá
610 Y=14
620 RETURN
630 ' Sol
640 Y=13
650 RETURN
660 ' Dó superior
670 Y=10
680 RETURN
690 ' Ré superior
700 Y=9
710 RETURN
720 ' Mi superior
730 Y=8
```

```
740 RETURN
750 ' Fá superior
760 Y=7
770 RETURN
780 '
790 ' VISOR
800 '
810 CLS
820 PRINT TAB (14);''PAUTA''
830 LOCATE 1,6
840 FOR T=1 TO 5
850 PRINT : PRINT STRING$(40,''—'');
860 NEXT
870 PRINT : PRINT
880 RETURN
890 ' GUARDAR NOTAS
900 '
910 DIM M(12)
920 FOR T=1 TO 12
930 READ M(T)
940 NEXT
950 DATA 478,426,379,358,319,284,253,239,213,190,179,159
960 RETURN
```

Até agora, todas as características sonoras discutidas se basearam na variação da duração e tonalidade das várias notas. O som resultante é semelhante ao que se ouviria de um pequeno órgão electrónico. No entanto, o seu Amstrad está equipado com potencialidades sonoras bastante superiores a estas — usando as instruções ENT e ENV, é possível criar todos os tipos de sons sintetizados.

A síntese sonora é de facto um tema bastante complicado, e seria impossível tratar todas as características do complexo circuito sonoro do Amstrad — pelo que nos concentraremos em apenas algumas das principais, aquelas que o leitor provavelmente estará mais interessado em usar de início.

Comecemos por um pouco de teoria. Considerando que conhece alguma coisa de música, por exemplo um pouco de notação musical, começaremos a investigar o canal de ruído com o objectivo de criar efeitos sonoros bastante realistas.

Primeiramente, o leitor deverá desligar o parâmetro tonal da instrução SOUND; isto permite-lhe aceder ao «ruído», em vez das notas musicais. O ruído pode ser usado para criar muitos efeitos realistas, desde tiros de armas de fogo e explosões até diferentes sons de instrumentos musicais. Existem cerca de quinze tipos de ruídos diferentes à disposição no Amstrad; cada um deles é considerado como uma *envolvente*.

A envolvente de ruído é o sétimo parâmetro da instrução SOUND. Para ouvir uma envolvente de ruído, escreva SOUND 1,0,20,6,0,0,N — onde N é um valor entre um e 15. Depois de ter experimentado diversos valores de N, poderá descobrir que as envolventes produzem sons bastante semelhantes, particularmente nos valores um a cinco. No entanto, quando as envolventes são usadas como base para um efeito sonoro, as diferenças são muito mais evidentes. Oiça o sétimo, oitavo e nono exemplos sonoros das páginas que se seguem se quiser escutar efeitos verdadeiramente interessantes de envolventes de ruído num ciclo FOR/NEXT.

A criação de sons mais complexos requer o uso de envolventes de volume e tom. A amplitude de um som dá-lhe um qualidade distintiva. Os sons que tratámos até agora possuem todos uma amplitude igual (exceptuando o programa de «desaparecimento» do som). Considere, por um momento, o som de uma nota de piano — a sua amplitude está longe de ser constante; observa-se um som claro quando se toca na tecla, o qual morre depois rapidamente. Uma guitarra eléctrica, por outro lado, aumenta mais lentamente e mantêm-se durante mais tempo. Podemos assim descobrir a diferença entre um vasto número de instrumentos e sons, apenas a partir das suas diferentes envolventes.

A variação do volume de um som pode ser representada num gráfico que crie uma «forma sonora». Tipicamente, uma envolvente pode ser dividida em quatro partes: o *ataque,* quando o som aumenta; *declínio,* quando o som perde alguma da sua amplitude; *sustentação,* quando a amplitude se mantém, e a *queda* da amplitude para zero. A figura 2 mostra o tipo de gráfico que podemos construir.

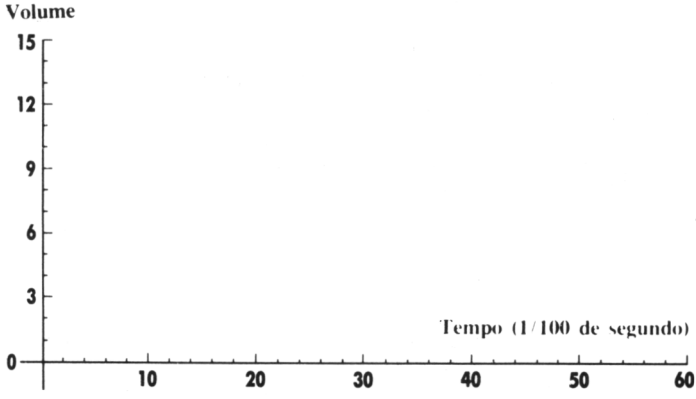


Fig. 2 — Um exemplo de envolvente de volume.

Esta envolvente pode ser controlada pelo comando ENV; esta instrução pode parecer muito confusa, podendo ter até 16 parâmetros, apesar de, de facto, não ser difícil de compreender. Vamos usar a instrução ENV para criar ondas sonoras com as quatro características de amplitude já mencionadas.

Em primeiro lugar teremos de dar a esta envolvente um número, de modo a podermos especificá-la num programa onde seja usada mais de uma instrução ENV. A seguir ao número da envolvente existem quatro partes da instrução ENV, uma para cada uma das características de amplitude. Cada secção possui três parâmetros: o número de passos de volume, a dimensão de cada passo e o tempo durante o qual se mantém cada passo (medido em centésimos de segundo). Nestas condições, é possível ter a seguinte instrução ENV:

ataque sustentação

ENV 1, 3, 5, 2, 7, −1, 1, 1, 0, 8, 4, −2, 4

n.º da envolvente declínio queda

Não esqueçamos que existem 15 passos de volume, não se podendo ultrapassar os valores máximo e mínimo. Para ajudar o leitor a conceber as suas próprias envolventes, a figura 3 mostra um gráfico onde se pode marcar a forma da onda.

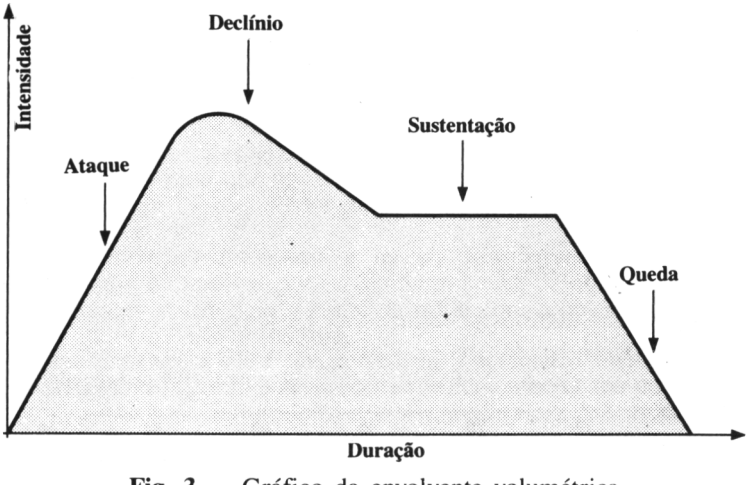


Fig. 3 — Gráfico da envolvente volumétrica.

Depois de criar uma envolvente, como pode incorporá-la no comando SOUND? Bem, é bastante simples dado que o quinto parâmetro da instrução SOUND é reservado ao comando ENV. Basta colocar neste lugar o número da envolvente; no exemplo já dado, o número da envolvente era 1. Nestas condições, uma instrução SOUND que chame a envolvente já definida poderia ser:

```
SOUND 1,150,30,12,1,0,0
```

Já discutimos para que serve o sétimo e último parâmetro da instrução SOUND. Resta apenas discutir o uso do sexto parâmetro, e para tal será necessário conhecer alguma coisa sobre as envolventes tonais e a instrução ENT.

A envolvente tonal é semelhante à volumétrica, mas não se utilizam as secções ataque, declínio, sustentação e queda na instrução correspondente.

A envolvente tonal tem a ver com o aumento e diminuição da tonalidade e é formada por um número de envolvente seguido por grupos de parâmetros, consistindo num certo número de passos, e pela dimensão e duração de cada um deles. É apenas necessário um grupo se a envolvente apenas aumenta ou diminui de tonalidade. Quando a envolvente aumenta e diminui, é necessário um grupo de parâmetros para cada variação. Um aumento e uma diminuição requerem dois grupos, enquanto dois aumentos e uma diminuição requerem três grupos. Isto pode ser visto muito mais claramente no exemplo seguinte:

```
10 ' Envolvente de amostra
20 '
30 ENT 1,25,-3,3
40 ' Envolvente número 1
50 ' 25 passos a -3 cada
60 ' durando 3/100 de segundo cada
70 '
80 SOUND 1,200,80,15,0,1
```

Se bem que o número de passos de volume esteja limitado a 15, existe uma flexibilidade muito maior quando se usa o comando ENT. Podem-se criar tantos passos quantos se quiserem, desde que o valor de tonalidade não exceda a vasta gama de instrução SOUND (um a mais de 4000). O exemplo anterior produz uma variação tonal num único sentido; o valor é negativo, indicando que a tonalidade aumenta (assim, a nota torna-se cada vez mais aguda).

Pode-se obter um efeito de *tremolo* fazendo a tonalidade aumentar e diminuir; isto é feito usando pelo menos dois grupos de parâmetros.

O curto programa que se segue cria um efeito de *tremolo* — escreva-o, execute-o e depois volte ao texto para compreender como funciona.

```
10 ' tremolo
20 '
30 ENT 1,12,-3,1,12,3,1
40 SOUND 1,250,200,15,0,1
```

Talvez o leitor fique bastante desiludido com o resultado final do programa; obteve apenas um pequeno *tremolo* seguido de uma nota. Há uma boa razão para isto — o comprimento da envolvente é apenas 24/100 de segundo, enquanto a nota tocada pela instrução SOUND dura 2,5 segundos. Se quiser que o *tremolo* dure tanto tempo quanto a nota, poderá actuar de duas maneiras. Uma consiste em acrescentar vários grupos de parâmetros, aumentando e diminuindo a tonalidade até atingir a duração da nota — haverá assim onze grupos de parâmetros. Obviamente isto não é prático; portanto, a BASIC Locomotive permite que, no caso de se pretender que a envolvente tonal dure tanto tempo como as notas tocadas (por outras palavras, se a duração da envolvente for menor do que a duração da nota, o computador repete a envolvente até ambas as durações serem iguais), basta acrescentar um sinal menos ao número da envolvente. Nestas condições, na rotina de *tremolo* indicada, a instrução ENT seria:

```
ENT -1,12,-3,1,12,3,1
```

A figura 4 mostra um gráfico para planeamento das envolventes tonais.

Vejamos alguns dos efeitos produzidos, quer usando a instrução ENT ou ENV, ou criando envolventes de ruído. Introduza na máquina cada um dos programas, e experimente alterar um valor aqui e outro ali. Talvez descubra exactamente o efeito sonoro que deseja!

```
10 ' EFEITO 1
20 '
30 ENT -1,5,5,1,10,-5,1,5,1,1
40 SOUND 1,500,10000,7,0,1
50 FOR T=1 TO 5000 : NEXT
60 PRINT CHR$(7)
```

```
10 ' EFEITO 2
20 '
30 ENT -2,5,-1,2,1,5,3
40 SOUND 1,100,400,14,0,2
50 SOUND 2,108,400,14,0,2
60 SOUND 4,92,400,14,0,2
```

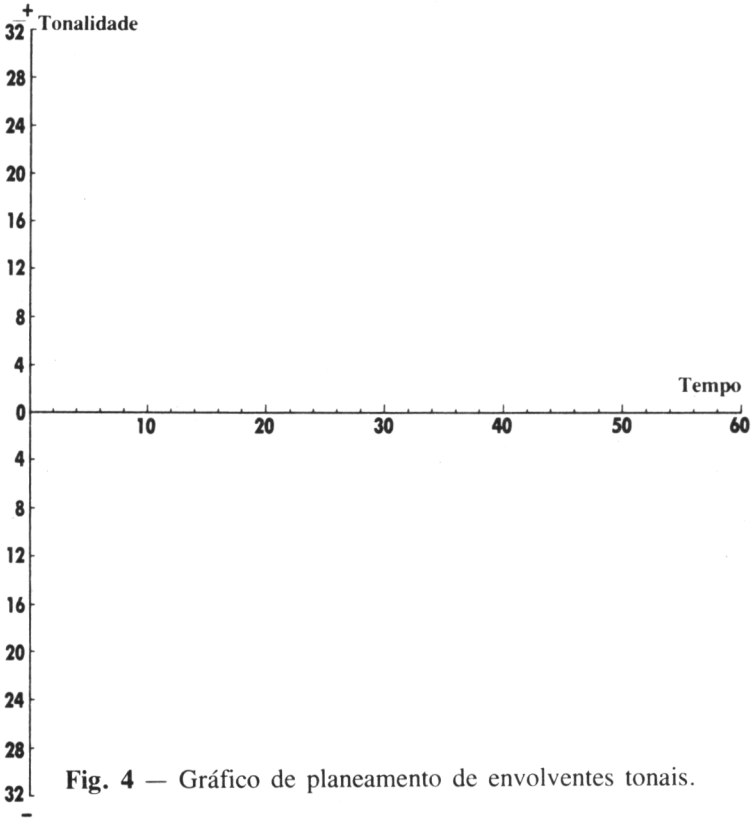


Fig. 4 — Gráfico de planeamento de envolventes tonais.

```
10 ' EFEITO 3
20 '
30 ENT -2,20,-1,1,1,20,1
40 SOUND 1,280,400,14,0,2

10 ' EFEITO 4
20 '
30 ENV 1,100,3,1
40 SOUND 1,200,100,1,1,1,5

50 GOTO 3010 ' EFEITO 5
20 '
30 ENV 1,40,6,2
40 SOUND 1,220,40,0,1,1,5
50 GOTO 30
```

```
10 ' EFEITO 6
20 '
30 ENT 1,10,-4,20
40 ENV 1,16,24,2
50 SOUND 1,150,100,1,1,1,7
60 WHILE INKEY$=" " : WEND
70 GOTO 40
```

```
10 ' EFEITO 7
20 '
30 FOR T=15 TO 1 STEP -1
40 SOUND 1,0,1,7,0,0,T
50 NEXT
60 GOTO 30
```

```
10 ' EFEITO 8
20 '
30 FOR T=15 TO 1 STEP -1
40 SOUND 1,0,7,5,0,0,T
50 SOUND 2,0,21,6,0,0,T
60 NEXT
70 GOTO 30
```

```
10 ' EFEITO 9
20 '
30 FOR T=15 TO 1 STEP -1
40 SOUND 1,0,7,5,0,0,T
50 SOUND 2,0,13,6,0,0,T
60 NEXT
70 GOTO 30
```

Sincronização

Antes de nos envolvermos excessivamente na sincronização, vejamos alguns programas que mostram os seus resultados. O maior dos dois programas apresentados reproduz uma melodia bem conhecida com um efeito de sintetizador, enquanto a rotina mais curta usa um efeito já conhecido, mas desenvolve-o usando três canais sonoros.

```
10 ' MÚSICA
20 '
30 ' SUBROTINA MUSICAL
40 '
50 GOSUB 300
60 ' TOCAR NOTAS
70 DIM P(100)
```

```
80 RESTORE 120
90 FOR T=1 TO 47
100 READ P(T)
110 NEXT
120 DATA 1,1,8,8,10,10,8,0,6,6,5,5,3,3,1
130 DATA 0,8,8,6,6,5,5,3,0,8,8,6,6,5,5,3
140 DATA 0,1,1,8,8,10,10,8,0,6,6,5,5,3,3,1
150 '
160 ' 0 = PAUSA
170 '
180 FOR T=1 TO 47
190 SOUND 1,M(P(T)),20,15
200 IF P(T) >2 THEN SOUND 2, (M(P(T)))-2,20,15 ELSE
    SOUND 2, M(P(T)),20,15
210 SOUND 4, (M(P(T)))+2,20,15
220 SOUND 1,0,10
230 SOUND 2,0,10
240 SOUND 4,0,10
250 '
260 ' SOUND 1,0 PRODUZ UM INTERVALO
270 '
280 NEXT
290 END
300 '
310 ' GUARDAR NOTAS
320 '
330 DIM M(24)
340 RESTORE 380
350 FOR T=1 TO 24
360 READ M(T)
370 NEXT
380 DATA 478, 451,426,402,379,358,338,319,301,284,268,253
390 DATA 239,225,213,201,190,179,169,159,150,142,134,127
400 RETURN
```

```
10 ' DESAPARECIMENTO DO SOM 2
20 '
30 FOR T=7 TO 1 STEP -1
40 SOUND 1,200+(5*T),20,T
50 SOUND 2,198+(5*T),20,T
60 SOUND 4,202-(5*T),20,T
70 NEXT
```

Esta sincronização dos três canais produz uma nota mais «espessa» e, quando os tons dos três canais são ligeiramente diferentes, o som resultante produz um efeito desfasado.

A sincronização pode ser usada para tocar uma melodia enquanto um outro canal (ou vários) toca um suporte rítmico. Apresenta-se em seguida um simples programa de exemplo; note como se usa a instrução EVERY para interromper o programa principal e levar o Amstrad a tocar um batimento baixo. Por que não modificar uma das melodias anteriormente apresentadas de modo a ser tocada com um ritmo?

```
10 ' SINCRONIZAR
20 '
30 Z=0
40 EVERY 20 GOSUB 90
50 FOR T=340 TO 80 STEP -20
60 SOUND 1,T,20,10
70 NEXT
80 GOTO 40
90 Z=Z+1 : IF Z=2 THEN Z=0 : GOTO 120
100 SOUND 2,1000,10,13
110 RETURN
120 SOUND 4,0,8,13,0,0,4
130 RETURN
```

Consegue-se uma sincronização complexa usando a instrução SQ. Esta cria uma «fita sonora» que permite organizar por ordem de execução as diversas notas.

Resumidamente, diremos que SQ(x) — onde x é o número do canal usado (qualquer dos canais um, dois ou quatro) — pode produzir um valor entre zero e 255, e este valor deve ser analisado para obter a informação necessária; cada um dos oito bits produz uma informação diferente (este aspecto é tratado com maior pormenor no manual do Amstrad, na secção intitulada A Sound Primer; basicamente, os bits indicam a situação da fita de sons, como, por exemplo, o canal sonoro que está a tocar e o número de espaços livres presentes, se os houver).

Para uma sincronização de canais diferentes atribuímos um valor especial ao canal SOUND; tratar-se-á de um dos cinco valores indicados em seguida:

8 sincronizar com o canal um
16 sincronizar com o canal dois
32 sincronizar com o canal quatro
64 manter o som que está a ser tocado actualmente
128 limpar as filas de sons

Para terminarmos este capítulo, vejamos um programa que exemplifica o modo de usar estes valores especiais do canal SOUND:

```
10 SOUND 10,300,100,15
20 WHILE INKEY$=" " : WEND
30 SOUND 17,380,100,15
```

Podemos verificar, ao executar o programa, que a primeira nota só é reproduzida depois de a segunda o ser, e que a segunda nota só é reproduzida depois de o utilizador carregar numa tecla.

O número de canal empregue na linha 10 é obtido usando o canal dois e sincronizando-o com o canal um (o que corresponde a um valor «especial» de oito; e portanto 8+2=10, que é o valor usado). O número de canal empregue na linha 10 é por sua vez obtido usando o canal um e sincronizando-o com o canal dois (usando o valor especial 16 — portanto, 16+1=17).

17

RUDIMENTOS DE CÓDIGO-MÁQUINA

Neste curto capítulo, tentaremos explicar o significado das instruções PEEK, POKE e CALL.

A PEEK e a POKE podem parecer mais estarrecedoras do que são na realidade. Vejamos, em termos simples, o que fazem. Imaginemos a memória do computador como uma pasta cheia de caixas de fósforos. Cada uma destas encontra-se numerada, de tal modo que podemos olhar para dentro de uma delas a ver o que se encontra no seu interior. Se não estiver lá nada, podemos pôr o que quisermos.

O número de caixa é conhecido por seu *endereço*, e para ver o que a caixa contém usamos a função PEEK. Para colocarmos um valor numa das caixas, usamos POKE. A seguir damos um exemplo do uso destas instruções. Escreva as palavras seguintes como comandos directos:

```
PRINT PEEK (32098)
POKE 32098,24
PRINT PEEK (32098)
```

A primeira ordem examina o endereço 32098 da memória do computador e descobre que nela está guardado um zero, ou seja, que a caixa está vazia.

A segunda ordem procura o endereço 32098 e deposita nele o valor 24; finalmente, a terceira ordem examina de novo o conteúdo desse endereço e descobre que nele se encontra agora o valor 24.

Usando as instruções PEEK e POKE podemos fazer muitas coisas, desde que conheçamos código-máquina e saibamos quais são os endereços que pretendemos alterar.

A instrução CALL é um pouco mais avançada do que a PEEK e a POKE, mas no essencial é muito semelhante à GOSUB, na medida em que chama uma subrotina; a única diferença é que a rotina chamada é um dos programas que constituem a ROM do Amstrad. Quando esti-

ver habituado a usar a instrução CALL, verá que CALL &BD19 é uma ordem extremamente útil, suavizando o movimento dos gráficos animados.

Para terminar, apresentamos um pequeno programa que serve para verificar o conteúdo da memória do computador entre os endereços zero e 600. Isto pode não parecer exactamente fantástico, mas torna-se certamente mais interessante se o leitor souber que o Amstrad guarda os programas BASIC a partir do endereço 368. Portanto, se executar este programa pode observar a forma como o programa BASIC se encontra guardado na memória do computador. O programa apresentado pode ser útil se for colocado no final de outro programa, e chamado quando se pretende por alguma razão verificar o conteúdo de quaisquer endereços.

```
10 ' PEEK
20 '
30 FOR T=0 TO 600
40 IF PEEK(T) > 31 THEN PRINT T, PEEK(T),CHR$ (PEEK(T))
50 IF PEEK(T) < = 31 THEN PRINT T, PEEK(T)
60 WHILE INKEY$=" " : WEND
70 NEXT
```

18

INTERRUPÇÕES DE SISTEMA

O seu Amstrad possui uma característica pouco habitual, a de permitir o uso de interrupções de sistema a partir da BASIC. Uma interrupção é exactamente aquilo que o nome sugere, um método de permitir ao computador sair do programa principal para realizar outras operações quase simultaneamente. Isto conduz a uma espécie de *multiprocessamento.*

Normalmente, o multiprocessamento só pode ser conseguido recorrendo ao uso de complexas técnicas em código-máquina. No entanto, o Amstrad possui duas instruções BASIC para este efeito, e alguns outros comandos que tratam os «relógios» da máquina que têm a ver com as interrupções.

A instrução AFTER

A instrução AFTER é usada no formato seguinte:

AFTER (tantos 1/50 de segundo), número do temporizador (entre 0 e 3), GOSUB (número de linha)

Expliquemos um pouco o que tudo isto quer dizer. Ao fim de um tempo previamente definido, a instrução AFTER dirige o computador para uma certa subrotina; o temporizador usado pode ser um dos quatro disponíveis no Amstrad (se estiver a usar várias interrupções num programa comprido, deverá ter cuidado em não as misturar, usando o temporizador errado).

Vejamos dois exemplos de programas utilizando interrupções: um deles recorrendo à AFTER (que já discutimos) e o outro empregando a instrução EVERY. Esta é seguida de dois números (tal como AFTER), que representam os mesmos elementos da instrução AFTER. A única diferença é que a interrupção é repetida sempre que é excedido o período de tempo especificado na primeira variável.

A primeira rotina é pouco mais do que uma demonstração do uso de AFTER. Note como, apesar de GOTO na linha 90, o programa salta para a subrotina especificada na instrução AFTER.

A segunda rotina é potencialmente muito mais útil, e o leitor poderá observar diversas variantes dela no Capítulo 16. O ciclo principal toca uma série de notas de tonalidade crescente. Depois de cada par de notas o programa passa à subrotina que toca na baixa, produzindo, portanto, um suporte grave. É óbvio que isto pode ser usado com grande efeito em rotinas musicais mais complicadas, dando um acompanhamento a três vozes à melodia básica. Por que não tenta simular isto num programa seu?

```
10 ' INTERRUPÇÕES 2
20 '
30 AFTER 150 GOSUB 100
40 ' ALTERAR FUNDO
50 PAPER 0
60 CLS
70 LOCATE 12,12
80 PRINT "O FUNDO É AZUL"
90 GOTO 90
100 ' ALTERAR FUNDO
110 PAPER 3
120 CLS
130 LOCATE 12,12
140 PRINT "O FUNDO É VERMELHO"
150 END
```

```
100 ' INTERRUPÇÕES
110 '
120 EVERY 40 GOSUB 180
130 FOR T=250 TO 70 STEP-15
140 SOUND 1,T,20,11
150 NEXT
160 FOR T=1 TO 2000 : NEXT : GOTO 130
170 '
180 ' TOCAR BAIXO
190 '
200 SOUND 2,500,20
210 RETURN
```

Podemos desactivar as interrupções com muita facilidade escrevendo simplesmente DI; para voltar a activá-las devemos escrever EI. Num programa musical onde, digamos, o computador toca uma nota baixa

por cada nota musical normal usando a instrução EVERY, será necessário desactivar as interrupções no final da peça tocada e durante as pausas compridas.

TIME

O Amstrad possui um relógio funcionando em tempo real, que pode ser acedido usando a instrução TIME. Se mandarmos a máquina imprimir TIME, obteremos no visor o tempo durante o qual o computador esteve ligado (em 1/300 de segundo). A partir deste valor, é fácil calcular o tempo de um jogo ou construir um relógio digital.

O valor de TIME ao fim de algum tempo de funcionamento do computador produzirá, no entanto, um valor extremamente alto — como poderemos usá-lo? Bem, se dividirmos esse valor por 300, obteremos um certo número de segundos (se dividirmos por 30 obteremos décimos de segundo). Por outro lado, na maior parte dos programas, o tempo deve ser inicializado; se a máquina estiver ligada durante meia hora antes de se começar a jogar, o jogador não gostará se, ao terminar o jogo em apenas um minuto, o computador lhe indicar que demorou 30 minutos e 57 segundos...

Não podemos passar TIME para zero — pelo menos em BASIC — e portanto como será possível conseguir isto? Começamos por igualar uma variável a TIME no início do jogo e depois, quando o jogo terminar, calculamos facilmente o tempo decorrido subtraindo ao valor actual de TIME o valor antigo. Isto pode parecer complexo, mas, se introduzir na máquina o pequeno jogo que se segue, compreenderá mais facilmente este princípio.

```
10 ' TESTE DE REACÇÕES
20 '
30 RANDOMIZE TIME
40 FOR T=1 TO 4 : D$=D$+CHR$(INT(RND*26)+65) : NEXT
50 FOR T=1 TO 1000 : NEXT
60 CLS : LOCATE 18,12
70 SOUND 1,180,40
80 START = TIME
90 PRINT D$
100 LOCATE 15,24
110 INPUT A$
120 IF A$=D$ THEN 150
130 GOTO 110
140 '
150 ' GANHAR
160 '
```

```
170 SCORE = TIME — START
180 CLS
190 LOCATE 10,10 : PRINT "TEM RAZÃO!"
200 LOCATE 7,12 : PRINT "E NECESSITOU DE"; SCORE/300;
    "SEGUNDOS"
210 LOCATE 10,20
220 PRINT "OUTRO JOGO?"
230 F$=INKEY$ : IF F$="S" THEN RUN
240 GOTO 230
```

Neste jogo é necessário escrever quatro letras e carregar em ENTER tão depressa quanto possível. O Amstrad indica-nos então quanto tempo demorámos (o seu melhor tempo foi 1,65 segundos — consegue ganhar-me?).

A temporização é tratada na linha 80, onde uma variável, START, é igualada a TIME imediatamente antes de as quatro letras serem impressas no visor. Assim que se introduz a resposta certa, o computador pára o relógio e torna outra variável, SCORE, igual ao tempo decorrido. O tempo de reacção guardado na variável SCORE é finalmente dividido por 300 antes de ser impresso no visor.

A última instrução que iremos examinar nesta secção não será a mais usada, mas é importante que se conheça a sua existência. A instrução REMAIN desactiva um temporizador se é usado numa interrupção (numa linha como EVERY 8,2 GOSUB 80 o temporizador usado é o 2) e imprime o número de 1/300 de segundo que restavam antes de o temporizador atingir o valor a que ocorre a interrupção (quando a instrução AFTER ou EVERY é executada). Assim, no exemplo acima, se o programa usasse REMAIN(2) (o valor entre parêntesis indica o temporizador) após terem passado apenas 5/300 de segundo, o computador desactivaria o temporizador dois e daria o valor três, porque restariam apenas 3/300 de segundo antes de a instrução EVERY ser executada, enviando o computador para a linha 80.

O programa de exemplo que se segue espera que o leitor carregue numa tecla e depois indica-lhe quanto tempo falta para satisfazer o próximo comando EVERY (ocorre uma interrupção sempre que soa a nota). Podemos usar este curto programa como um temporizador de reacções, tentando medir até que ponto nos conseguimos aproximar de 100 ou de qualquer outro número na gama 1 a 100 da instrução EVERY.

```
10 ' REMAIN
20 '
30 EVERY 100,3 GOSUB 70
40 WHILE INKEY$=" " : WEND
```

```
50 PRINT REMAIN(3)
60 END
70 SOUND 1,100,10,15
80 RETURN
```

19

JANELAS

O Amstrad possui a característica bastante útil que consiste em permitir a criação de janelas; estas assemelham-se a visores no interior do visor. A instrução que define uma janela é a seguinte:

WINDOW # A,X,X1,Y,Y1

Como já se referiu, o Amstrad possui um certo número de saídas de *stream*. Os *streams* zero e sete são usados para saídas de texto para o visor sob a forma de janelas, enquanto o *stream* oito corresponde à saída para a impressora e o *stream* nove à saída para a *cassette*. O valor A é o número do *stream*, enquanto X e Y são as coordenadas da janela; X e X1 são os valores na horizontal e Y e Y1 os valores na vertical. Portanto, (X,Y) representa as coordenadas do canto superior esquerdo da janela e (X1, Y1) definem o canto inferior direito.

As instruções CLS, PAPER e PEN trabalham com janelas. Pode-se definir a cor de fundo de uma janela usando PAPER # A,C, onde A é o número de *stream* e C a cor. PEN # A,C e CLS # A funcionam do mesmo modo que a instrução de PAPER, se bem que CLS não indique uma cor — a janela passa inteiramente ao valor de PAPER existente.

O programa que se segue define sete janelas e preenche-se com diferentes cores:

```
10 ' JANELAS POR TODO O LADO
20 '
30 INK 0,0 : MODE 0 : CLS
40 PRINT "JANELAS POR TODO O LADO"
50 INK 8,6 : INK 14,24 : INK 6,7
60 BORDER 1
70 WINDOW # 1,1,5,3,7
80 WINDOW # 2,8,12,3,7
90 WINDOW # 3,15,19,3,7
```

```
100 WINDOW # 4,1,5,10,14
110 WINDOW # 5,8,12,10,14
120 WINDOW # 6,15,19,10,14
130 WINDOW # 7,8,12,17,21
140 FOR T=1 TO 7
150 PAPER # T, (T*2)
160 CLS # T
170 PEN # T,0
180 PRINT # T," "
190 PRINT # T, "No.";T
200 NEXT
210 WHILE INKEY$=" " : WEND
220 MODE 1
230 END
```

O programa é bastante simples — as linhas 70 a 130 definem as janelas, enquanto as linhas 140 a 200 controem um ciclo que determina a cor de papel, limpa a janela e imprime o número da janela a tinta negra.

Existem alguns outros pontos que convém referir aqui. Em primeiro lugar, pode-se listar um programa numa janela apenas — experimente listar o programa anterior na janela sete. Isto consegue-se muito simplesmente escrevendo LIST 7. Em segundo lugar, convirá recordar que cada janela nova se sobrepõe a uma antiga. Portanto, se a última janela tinha a dimensão de todo o visor, esconde todas as janelas anteriores.

Existe uma outra ordem que está especificamente relacionada com as janelas, a instrução WINDOW SWAP. Este comando permite trocar o texto de uma janela com o de outra. O programa que se segue demonstra isto; a demonstração em causa é muito semelhante ao programa anterior, exceptuando o facto de usar esta nova instrução para deslocar o conteúdo de cada janela. O formato da instrução considerada é o seguinte:

```
WINDOW SWAP X,Y
```

onde X designa um *stream* e Y outro — o texto e os pormenores relativos à janela são transferidos entre as duas janelas. Experimente o programa que se segue:

```
10 ' TROCA DE JANELAS
20 '
30 INK 0,0 : MODE 0 : CLS
40 PRINT "TROCA DE JANELAS"
50 INK 8,6 : INK 14,24 : INK 6,7
```

```
60 RANDOMIZE TIME
70 BORDER 1
80 WINDOW # 1,1,5,3,7
90 WINDOW # 2,8,12,3,7
100 WINDOW # 3,15,19,3,7
110 WINDOW # 4,1,5,10,14
120 WINDOW # 5,8,12,10,14
130 WINDOW # 6,15,19,10,14
140 WINDOW # 7,8,12,17,21
150 WHILE INKEY$=" "
160 FOR T=1 TO 7
170 PAPER # T,(T*2)
180 CLS # T
190 PEN # T,0
200 PRINT # T," "
210 PRINT # T," No. ";T
220 NEXT
230 A=INT(RND*7)+1
240 B=INT(RND*7)+1
250 IF A=B THEN 230
260 WINDOW SWAP A,B
270 WEND
280 MODE 1
```

Uma outra forma de usar as janelas é para criar gráficos. A janela colorida é de longe o método mais rápido e simples de preencher um quadrado ou rectângulo. Este princípio foi levado um pouco mais longe na rotina gráfica que se segue; as janelas cujas dimensões aumentam e diminuem rapidamente produzem o efeito observado no visor durante a sua execução.

```
10 ' EFEITOS GRÁFICOS USANDO JANELAS
20 '
30 MODE 2
40 X=40 : Y=13
50 B=0
60 INK 1,6 : INK 0,1
70 PAPER # 1,3
80 FOR A=1 TO 40
90 IF A/3=INT(A/3) AND A >5 THEN B=B+1
100 IF X-A=0 THEN X=X+1
120 CLS # 1
130 SOUND 1,450-A*10,10
140 NEXT A
```

```
150 FOR T=1 TO 1500
160 NEXT T
170 PAPER 1
180 CLS
190 PAPER # 1,0
200 FOR A=40 TO 1 STEP -1
210 IF A/3=INT(A/3) THEN B=B-1
220 IF A/2=INT(A/2) THEN J=J+1 : IF J=2 THEN J=0
230 WINDOW # 1,X-A,W+A,Y-B,Y+B
240 PAPER # 1,J
250 CLS # 1
260 SOUND 1,450-A*10,10
270 NEXT A
280 FOR T=1 TO 1500
290 NEXT T
300 ON BREAK GOSUB 320
310 GOTO 70
320 MODE 1
330 END
```

20

AS INSTRUÇÕES *ON ERROR* e *ON BREAK*

A BASIC Locomotive inclui duas instruções menos vulgares: ON BREAK GOSUB e ON ERROR GOTO. Depois de cada um destes comandos, deve existir um número correspondente à linha para onde o programa salta se a condição for satisfeita (se a tecla BREAK está premida ou ocorrer um erro).

Existem diversos outros comandos que o leitor deve conhecer, como ERR e ERL; são variáveis usadas nas mensagens de erro — ERL é o número de linha onde foi criado o último erro, enquanto ERR é o tipo de mensagem de erro. Num dos apêndices apresenta-se uma lista das mensagens de erro.

O comando ERROR é seguido de um número (não entre parêntesis) correspondente ao número de erro. Se tal comando for executado, o computador realizará a acção apropriada. Assim, se se escrever ERROR 8, o computador imprimirá "Line not found" («linha não existente»).

O comando ON BREAK STOP é o último deste grupo de comandos, e serve para parar o programa quando se acciona a tecla BREAK. A seguir apresentam-se vários exemplos do uso destes comandos:

```
10 ' ON BREAK
20 '
30 ON BREAK GOSUB 80
40 FOR T=1 TO 100
50 PRINT T
60 NEXT T
70 END
80 RETURN
```

```
10 ' ON ERROR
20 '
30 ON ERROR GOTO 90
40 FOR T=1 TO 12
```

```
50 PRINT A(T)
60 NEXT
70 END
80 '
90 ' IMPRIMIR TIPO DE ERRO, NÚMERO E FIM
100 '
110 PRINT : PRINT ERR;" EM ";ERL
120 PRINT
130 GOTO 70
```

```
10 ' ON BREAK STOP
20 '
30 J=1
40 ON BREAK GOSUB 60
50 FOR T=1 TO 10000 : NEXT
60 J= J+1 : IF J=3 THEN 90
70 RETURN
80 FOR T=1 TO 10000 : NEXT
90 ON BREAK STOP
```

O primeiro programa impede-nos de interromper a sequência de números, entre um e 100, que está a ser impressa no visor. Isto pode ser facilmente feito nos nossos programas, talvez acrescentando um comentário dirigido ao utilizador e indicando-lhe que não deve tentar parar o programa...

O segundo programa contém um erro deliberado: é impresso no visor um *array* que não foi dimensionado. Não há problema até o computador tentar imprimir o décimo primeiro elemento do *array*; indica então um erro. Em vez de imprimir o erro em inglês e parar o programa, o computador imprime o número do erro usando ERR e a linha onde ocorreu usando ERL.

O terceiro programa constrói uma situação em que é necessário carregar na tecla de paragem (ESC) quatro vezes antes de o programa ser interrompido. De cada vez que se carrega na tecla em causa é somado um à variável J. Quando J chega a três (quando a tecla já foi premida três vezes), o programa salta para a linha 90, onde, se a tecla for novamente premida, o programa pára.

Quando for detectado um erro pelo comando ON ERROR GOTO, o programa pode ser continuado usando uma das seguintes três formas do comando RESUME:

RESUME isolado — continua a partir da linha onde se encontrou o erro.

RESUME seguido de um número de linha — continua a partir do número de linha especificado.

RESUME NEXT — continua a partir da instrução que se segue imediatamente à que originou a situação de erro.

Se modificarmos o programa de demonstração de ON ERROR já dado, de tal modo que a linha 130 contenha RESUME 40, é possível ver a diferença produzida por esta nova instrução. Assim que é encontrado um erro no programa, o ciclo FOR/NEXT inicia-se de novo.

Todas estas instruções são úteis quando protegemos os nossos programas e quando testamos programas de desenvolvimento.

21

INSTRUÇÕES DIVERSAS

A gama de instruções existentes num computador desafia muitas vezes qualquer tentativa de classificação, tornando necessário um capítulo como este. Discutimos aqui alguns comandos que, apesar de muito importantes, não se enquadram bem em qualquer das outras secções.

Já encontrámos a função FRE, que usámos para determinar a quantidade de memória disponível. Mas esta instrução tem outra utilidade, a de «recolha de lixo»: elimina variáveis de cadeia que tendem a encher a memória. O que acontece é que, de cada vez que se usa uma cadeia, esta é novamente guardada em memória — portanto, pode ocupar 10 K ou mesmo 20 K se é tratada por um ciclo FOR/NEXT, tal que a cadeia é alterada em cada ciclo. E, como é óbvio, o Amstrad só está interessado na última versão da cadeia.

Usando a instrução FRE (" ") é possível contrariar este enorme desperdício de memória. O programa que se segue mostra FRE (" ") em acção e demonstra claramente o gasto de memória devido à redefinição constante de uma cadeia:

```
10 ' USO DE FRE PARA LIBERTAR MEMÓRIA
20 '
30 CLS
40 PRINT "MEMÓRIA ORIGINALMENTE LIVRE:"; FRE(0)
50 LOCATE 3,12
60 PRINT "MEMÓRIA ACTUALMENTE DISPONÍVEL:"
70 FOR T=1 TO 200
80 A$=A$+CHR$(INT(RND*26)+65)
90 LOCATE 30,12
100 PRINT FRE(0)
110 NEXT
120 LOCATE 2,18
130 PRINT "MEMÓRIA LIVRE APÓS USO DE FRE( ): ";FRE(" ")
```

A instrução FRE(" ") é invocada sempre que se encontra numa linha de programa, mas o melhor modo consiste em atribuir FRE("#") a uma variável indesejada em diversos pontos do programa. O programa acima imprime FRE(" "), mas o leitor pode não querer que a quantidade de memória livre apareça misteriosamente no visor. É consigo...

Também ligado aos usos da memória, a instrução HIMEM indica o endereço de memória do maior byte usado em BASIC, e pode ser usada em expressões numéricas da forma habitual. Quando a máquina é ligada, o valor HIMEM é geralmente 43903.

A instrução MEMORY é também usada para efeitos de memória (quem diria?). Define o endereço mais alto que pode ser usado em BASIC; de facto, define o valor de HIMEM. O principal uso desta instrução consiste em reservar memória de modo a não poder ser usada em BASIC, permitindo guardar rotinas em código-máquina.

O Amstrad e uma impressora

Existem três instruções BASIC que é necessário conhecer quando se usa uma impressora com o Amstrad: WIDTH, PRINT # 8 e LIST # 8.

A instrução WIDTH controla simplesmente a largura da cópia impressa e é muito útil. A seguir à instrução WIDTH deve encontrar-se um único valor, a saber, o número de caracteres que pretendemos ver impressos em cada linha.

PRINT # 8 é semelhante à instrução PRINT vulgar, exceptuando o facto de a saída ser feita para a impressora e não para o visor. Portanto, a seguir à vírgula, pode-se acrescentar TAB, pontos e vírgulas, vírgulas, e a maior parte das outras instruções usadas com PRINT.

LIST # 8 é bastante semelhante à LIST normal. Se quisermos listar porções de um programa, usamos o formato:

LIST número de linha — número de linha, # 8.

Esta instrução imprime as linhas entre os números especificados (inclusive), na impressora.

No que se refere aos códigos da impressora usados para criar estilos de letra especiais, sublinhados, etc., todo o trabalho do utilizador se resume em grande parte a procurar os códigos no manual da impressora. Todos os códigos que usei neste livro se aplicam simultaneamente às impressoras Epson e Star — no momento em que escrevo, as impressoras mais populares em uso com o Amstrad (não falando na do próprio Amstrad).

22

O AMSTRAD CPC 6128

Lançado apenas cerca de um ano depois do primeiro modelo da Amstrad, o CPC 6128 foi originalmente vendido apenas nos Estados Unidos.

A máquina possui 128 K de memória directa em vez dos 64 K do 464 e do 664, um *drive* de disco de 3 polegadas, uma versão mais avançada do CP/M e um teclado redesenhado.

A RAM é igual à existente nas outras máquinas Amstrad. Baseia-se no circuito integrado Z80 e utiliza exactamente o mesmo dialecto BASIC existente no CPC 464 e no CPC 664. As novas instruções para acesso à memória extra e para uso do disco encontram-se todas guardadas neste. Isto significa que a maior parte do *software* existente para as duas máquinas anteriores é compatível com a nova máquina. Nestas condições, os programas apresentados neste livro podem ser executados em qualquer dos três modelos.

O teclado do 6128 possui todas as teclas existentes nas duas máquinas anteriores, sendo a única diferença a dimensão e a posição de muitas das teclas.

A duplicação da memória coloca problemas no caso de uma máquina baseada no Z80. Este circuito só pode endereçar 64 K, sendo necessário recorrer a truques para poder aceder ao segundo bloco de 64 K. O Amstrad utiliza este segundo bloco para guardar imagens e construir ficheiros de dados em RAM, mas não para escrita de programas. Isto não constitui uma grande desvantagem. Quaisquer programas por nós escritos que atinjam os limites da memória utilizável para tal (cerca de 43 K) contêm provavelmente uma grande quantidade de dados. Um grande programa de aventuras necessitará de uma substancial quantidade de memória para armazenar os pormenores relativos a cada local, objecto e comando. Um programa de contabilidade terá de guardar todas as transacções relativas a cada conta. Estes dados podem ser guardados sob a forma de um ficheiro de dados no interior da memória, em vez de estarem guardados em *cassette* ou disco.

As instruções para acesso à memória extra sob a forma de ficheiro de dados em RAM são :BANKOPEN, :BANKWRITE, :BANKREAD e :BANKFIND.

:BANKOPEN inicializa o número de registo e define o comprimento do registo. :BANKWRITE e :BANKREAD escrevem e lêem os dados no ficheiro. :BANKFIND é uma instrução de procura muito útil. Produz o número do registo onde se encontra guardado o dado especificado.

O bloco de memória extra pode ser igualmente usado para guardar imagens. Quando usado deste modo, fica dividido em quatro secções de 16 K cada, cada uma delas capaz de guardar uma imagem. Os comandos :SCREENCOPY e :SCREENSWAP deslocam as imagens. Para apresentar uma delas, realiza-se um :SCREENSWAP que desloca a imagem para o primeiro bloco, que corresponde ao ficheiro de imagem. Isto apresenta óbvias potencialidades em programas de jogo, permitindo a passagem rápida de um cenário a outro, e também em aventuras, onde é possível apresentar cenas complicadas rapidamente enquanto outras imagens são construídas para serem apresentadas mais tarde.

O CP/M pode ser usado no CPC 664 e no 464 com *drive* de disco, e é tratado num capítulo anterior. A versão do CP/M usada nestas máquinas é a 2.2, talvez a mais popular, mas que conduz a um problema quando usada no Amstrad. O CP/M necessita de 64 K em RAM, e ambas estas máquinas não podem fornecer todo esse espaço livre. Isto significa que muitos dos programas disponíveis em CP/M devem ser alterados ou reescritos para poderem funcionar no Amstrad. O 6128 não tem este problema. Utiliza uma nova versão do CP/M, a versão 3.0. Esta posssui a vantagem de poder utilizar sistemas de comutação de memórias, podendo portanto recorrer aos 64 K de memória extra existente neste computador. Portanto, a maior parte dos programas CP/M podem correr sem alteração no 6128. Convém apenas notar um ponto: a maior parte das máquinas de CP/M utilizam discos de 5¼ polegadas, pelo que é necessário dispor de versões gravadas em discos de 3 polegadas. Isto não deverá ser um grande problema.

23

COMO ESCREVER PROGRAMAS DE UMA FORMA MAIS PROFISSIONAL

Alguns truques de programação

1) Como posso desactivar a tecla ESC para que as outras pessoas não entrem nos meus programas?

Existem várias maneiras de o fazer. Um método consiste em redefinir a tecla ESC usando a instrução KEY DEF; por exemplo KEY DEF 66,1,32 leva-a a imprimir um espaço. Em segundo lugar, pode-se usar ON BREAK GOSUB para obrigar a máquina a saltar para a subrotina que imprime uma mensagem de aviso e depois volta ao ponto onde a execução se encontrava sem parar o programa. Em terceiro lugar, pode-se usar um valor de CALL para desactivar esta tecla — experimente CALL 47944. Para activá-la de novo, deve-se escrever CALL 47947. Ambas estas instruções só funcionam em programa, pelo que devem ser inscritas nele.

2) Como posso impedir que um programa seja apagado acidentalmente (com NEW)?

Bom, POKE 48622, 201 desactiva a função CTRL-SHIFT-ESC que limpa a memória do computador. Para activar de novo esta função, escreva POKE 48622,195.

O método citado evitará que alguém perca um programa, mas, se tem medo de que alguém escreva DELETE seguido de uma parte significativa do programa ou escreva NEW e depois ENTER, necessitará de ir um pouco mais longe. A escrita destes comandos exige que o computador se encontre em modo directo, pelo que, se for possível, mantenha o programa em execução e use os truques referidos na primeira questão para impedir que alguém o pare. No entanto, alguns programas podem não permitir isto; se for este o caso, sugiro que, no caso de não necessitar da letra E, a redefina para outra letra, criando assim um erro de sintaxe quando o utilizador escreve NEW ou DELETE.

Não se esqueça de que, como programador, se utilizar qualquer destas soluções para os problemas citados, deve deixar a si mesmo uma «porta de saída» — de tal modo que, ao desenvolver o programa, não se veja forçado a desligar o Amstrad e começar de novo.

3) Existe alguma forma simples de impedir que os outros observem o meu programa?

Se não pretende blocos selectivos sobre certas funções, mas sim impedir qualquer forma de listar o seu programa, deve utilizar o modo protegido da instrução de SAVE: SAVE "Nome",P impedirá a listagem.

4) Quero acrescentar ao meu programa um modo de medir a mais alta pontuação — que posso fazer?

Existem várias maneiras de criar esta função, usando a instrução MAX para consultar todas as listas de pontuações até encontrar a mais elevada.

Se pretende criar um sistema mais complexo, será conveniente utilizar uma ordenação numérica, enviando depois a informação para uma tabela. Observe o programa que se segue:

```
10 ' TABELA DE PONTUAÇÕES
20 INK 2,1,24
30 DIM H$(8,2)
40 ' LEITURA DE PONTUAÇÕES FALSAS PARA O ARRAY H$
50 FOR T=1 TO 8
60 READ H$(T,1) : READ H$(T,2)
70 NEXT
80 DATA ROBERT,1200,FRANK,990,GEORGE,2340
90 DATA ARNOLD,9910,MARK,780,TIM,1500
100 DATA SCOTT, 2200,MARY,1760
110 ' APRESENTAR PONTUAÇÕES
120 CLS
130 LOCATE 12,4
140 PEN 3
150 PRINT ''TABELA DE PONTUAÇÕES''
160 PEN 1
170 PRINT TAB(10); ''=====================''
180 FOR T=1 TO 8
190 IF T=1 THEN PEN 2 ELSE PEN 1
200 PRINT TAB(10);''|'';TAB(29);''|''
210 PRINT TAB(10);''|'';TAB(12);H$(T,1);
    TAB(24);H$(T,2);TAB(29);''|''
220 NEXT
230 PRINT TAB(10);''|'';TAB(29);''|''
240 PRINT TAB(10); ''=====================''
250 SOUND 1,180
```

5) Existe alguma maneira de usar o botão de movimento do meu gravador para localizar um programa sem necessidade de colocar o computador em modo LOAD, SAVE ou CAT?

Quando se posiciona a fita à procura de um dado programa, as teclas de rebobinação podem ser demasiado rápidas. O uso da tecla normal de movimento permite controlar muito melhor a posição. Para usar esta tecla quando não se utilizam os comandos normais de tratamento de *cassettes* em computador, deve-se escrever OUT 512,16. Para desactivar novamente a tecla do gravador, escreve-se OUT 512,0.

O meu último conselho àqueles que tentam escrever um programa de jogo de algum tipo é que se concentrem na sua originalidade. Existem muitas cópias de jogos clássicos, e a criação de um original é muito mais interessante e, no caso de se desejar qualquer lucro comercial, mais remunerador do que o desenvolvimento de uma cópia. De facto, não há qualquer interesse em criar um jogo novo, usando efeitos magníficos, se o resultado é posto de parte ao fim de pouco tempo.

AMSTRAD PCW8256/8512

O Amstrad PCW8256 foi apresentado em 1985 como «uma máquina de escritório», destinada a substituir as máquinas de escrever electrónicas. Na verdade era um computador de 8 bits, equipado com o microprocessador Z80, trabalhando à velocidade invulgar de 8 Mhz, o que tornava o PCW mais rápido que alguns computadores de 16 bits. A maior novidade quanto ao PCW foi, todavia, o facto de ser o primeiro conjunto integrando a unidade central, o monitor, o sistema de disquetes e a impressora. Isso, aliado à presença de um excelente programa de processamento de texto — o «Locoscript» — e também do CP/M Plus, dando acesso ao poderoso Mallard Basic, à última versão do DR LOGO e (através da extensão GSX) aos programas de gráficos, tudo por um preço muito inferior ao dos componentes dos sistemas do mesmo tipo então disponíveis, conduziu a um tremendo sucesso, tendo as vendas do PCW8256 ultrapassado rapidamente as de todos os outros produtos da Amstrad.

O PCW possui um teclado profissional com teclas numéricas, cursoras, e de funções, dando acesso aos caracteres acentuados das principais línguas — incluindo a portuguesa —, com a mesma ou maior facilidade que o de uma máquina de escrever, e também aos caracteres gregos e praticamente a todos os símbolos correntes na contabilidade. O monitor, de fósforo verde, é excepcionalmente grande e nítido, comportando 90 colunas, mas podendo funcionar a 80 em CP/M. O leitor de disquetes de 3 polegadas e 360 Kb (2 × 173 Kb) está alojado na caixa do monitor, que comporta também a unidade processadora central, com 256 Kb de RAM, sendo 102 Kb utilizáveis no «disco de memória» que facilita o tratamento de dados extensos. A impressora é directa e automaticamente comandada pelo computador, com carregamento também automático, dando o «Locoscript» acesso a espaçamentos de 10, 12, 15 e 17 caracteres por polegada e proporcional, de largura simples ou dupla, altura normal ou meia, expoente ou índice, redondo ou itálico, com entrelinhas de 0 a 3, passo variável entre linhas, justificação à esquerda e à direita, tabulação automática e impressão em *draft* (rápida), «NQL» (qualidade correio), *Bold* e *Double Strike*. As cópias de ecrã são obtidas pela pressão simultânea em três teclas — SHIFT, EXTRA e PRINT.

O «Locoscript» dá acesso directo a todos os comandos por combinação de teclas ou menus e permite a inserção de textos, a busca e substituição de palavras ou frases, a gravação de trechos completos na memória e sua subsequente inserção, a inserção de resumos dos textos, para reconhecimento rápido destes, a edição de fichas ASCII, etc., e permite classificar os textos conforme as conveniências do utilizador, fornecendo automaticamente o directório das disquetes com essa organização, o espaço ocupado por cada ficha, o espaço disponível na disquete e o espaço ocupado no disco de memória.

Uma das características mais importantes do «Locoscript» é a presença de um grande número de matrizes que permitem a execução rápida de correspondência, avisos, anúncios, relatórios, e até publicações, com organização de capítulos, títulos em cabeça ou rodapé, numeração automática e marginação par-ímpar, etc. O único inconveniente é a lentidão das operações que implicam o correr dos textos, principalmente quando estes são longos.

O CP/M Plus (ou CP/M 3.0) é excepcionamente rico em utilidades e em extensões. O Mallard Basic, acessível através do CP/M, é muito rápido, tem uma alta precisão e permite a organização de ficheiros indexados através do sistema Jetsam. Tem como único inconveniente o facto de ser algo complexo na manipulação do ecrã, mas isso pode ser contornado definindo funções a teclas. Carece de capacidade gráfica, mas esta pode ser obtida através da extensão GSX e de programas comerciais como o DR GRAPH (gestão) e o DR DRAW (desenho geral). O DR LOGO, fornecido com PCW, permite criar gráficos geométricos e é muito flexível, sendo particularmente útil para a manipulação de listas.

Compreende-se que os utilizadores do PCW tenham descoberto bem depressa que ele era muito mais do que um processador de texto e que o mesmo tenha acontecido com os produtores de *software*. Versões dos principais programas comerciais, como o «Supercalc2», o «DatabaseII», o «Multiplan», o «Cardbox», etc., foram transferidos para o PCW, que se tornou num dos computadores com maior amplitude de opções quanto a bases de dados, folhas de cálculo, ficheiros, etc. — incluindo processadores de texto alternativos, como o «Wordstar» e o «Tasword 8000». Isso expandiu ainda mais os seus usos e levou em 1986 ao aparecimento da versão 8512, com um segundo leitor de disquetes de 1Mb (706 Kb utilizáveis) e um disco de memória de 368 Kb. Os produtores independentes de *hardware* deram também conta desse interesse e têm apresentado um número surpreendente de periféricos complementares, desde *modems* a canetas de luz e pranchetas de gráficos de alta qualidade, e a leitores de disquetes de 5,25 polegadas, conjugados com programas que permitem a transferência de fichas IBM PC e vice-versa.

As aplicações do PCW estenderam-se até ao campo dos jogos, com a adaptação de programas como o «Batman», o «Fairlight», etc. — a alta definição do ecrã compensando a falta de cor. Também nesse campo se tem multiplicado o aparecimento de periféricos, desde *joysticks* a ... sistemas de música — o que é particularmente curioso porque os PCW, originalmente, dispõem apenas de uma campainha de aviso.

Com o aparecimento do Amstrad PC1512, surgiram dúvidas sobre o futuro dos PCW — dúvidas logo afastadas pela Amstrad, porquanto os dois tipos de máquinas são na realidade muito diferentes e dedicados a públicos distintos. Os PCW são particularmente úteis aos profissionais e às pequenas empresas, enquanto os PC são mais aptos às aplicações de maior porte e em particular ao uso em redes, em empresas médias.

*

AMSTRAD PC1512

A apresentação do Amstrad PC1512, em Outubro de 1986, foi saudada como o início de uma nova era na informática, pois tratava-se de um computador altamente compatível com a norma estabelecida pelo IBM PC, mas com características de nível superior, por preços da ordem da terça ou quarta parte das máquinas da mesma classe. Compreendendo que o PC1512 iria ter grande procura, as principais casas de *software* de gestão baixaram os preços dos programas mais vulgarizados, em alguns casos para perto de um décimo. O acolhimento que consequentemente teve o novo Amstrad PC, afastando do mercado todos os «compatíveis» de origem duvidosa e comprometendo as vendas dos modelos comparáveis das marcas há mais tempo estabelecidas, originou uma campanha de rumores cuja falsidade foi bem depressa demonstrada.

O Amstrad PC1512 é na realidade uma excelente máquina, construída à base do microprocessador 8086 de 16 bits e funcionando à velocidade de 8 Mhz — muito mais rápido que o IBM PC básico e que a grande maioria dos «compatíveis» de baixo preço. Quando utiliza o Basic 2 — especialmente desenvolvido para ele e incluído no equipamento original —, a sua rapidez aproxima-se da do IBM PC-AT. Dispõe de 512 Kb de RAM, expansíveis a 640 Kb, parte da qual configurável em «RAM disc», e de um ou dois leitores de disquetes de 5,25 polegadas e 360 Kb. O segundo leitor pode ser substituído por um disco rígido de 10 ou 20 Mb. Foi também previsto o espaço para três placas de expansão normais, de muito fácil montagem, para *modems*, redes locais, discos rígidos suplementares, etc., não sendo necessário mais porque

muitas das funções das placas correntes — por exemplo, as de cor e gráficos — estão incorporadas na placa principal, que permite a inclusão do processador matemático 8087.

O Amstrad PC pode ser fornecido com monitor a cores ou monocromático, anti-reflexos, sendo neste caso as cores substituídas por graduações de cinzento. Possui cinco modos de apresentação: 40×25 caracteres com 16 corpos, 80×25 com 6 cores, gráficos de 320×200 pontos com 3 paletas de 4 cores ou 640×200 com 2 cores, ou ainda alta resolução especial com 640×200 com 16 cores. Dispõe de altifalante regulável. A fonte de alimentação está incorporada no monitor, reduzindo ao mínimo o número de cabos sobre a mesa de trabalho, como é norma da Amstrad. Uma *interface* série RS232 permite a ligação de impressoras, *modems* e outros acessórios. Uma *interface* paralela permite a ligação de impressoras do tipo Centronica, nomeadamente a Amstrad DPM300, especialmente concebida para o PC1512. Uma ficha do tipo mini-D permite a ligação de um «rato», fornecido com o computador. Pode também funcionar com *joystick* e caneta de luz. O teclado, de 85 teclas alfanuméricas, numéricas e de funções, tem altura regulável.

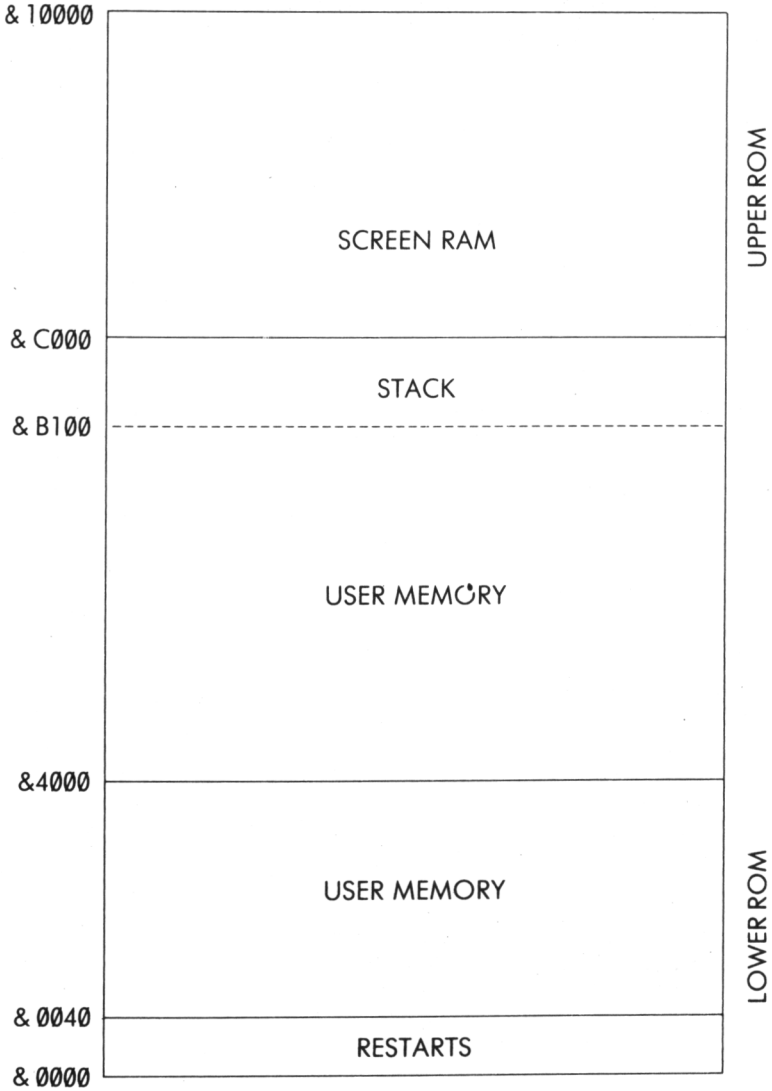
Com o PC1512 são fornecidas quatro disquetes com os sistemas operativos MS-DOS 3.2, da Microsoft, e DOS PLUS, da Digital Research, o GEM (Graphics Environement Manager), também da DR, e o BASIC 2, da Locomotive, que usa o GEM. O MS-DOS 3.2 inclui diversos utilitários, nomeadamente os que permitem a gestão do «disc RAM» e «Spooling», e a ligação a discos rígidos e em rede. O DOS PLUS pode funcionar com programas em MS-DOS e CP/M 86 e permite a utilização simultânea por vários utilizadores, em várias tarefas. O GEM apresenta a informação sob a forma de ícones e em janelas com menus deslizantes, permitindo que os comandos sejam feitos pelo «rato», sem necessidade de premir as teclas. Inclui o GEM Desktop, com relógio e calculadora (o PC1512 possui relógio com bateria de salvaguarda), e o GEM PAINT, para gráficos e desenhos. O GEM pode ainda funcionar com programas como os GEM Diary (incluindo o Cardbox), Graph, Wordchart, Draw e Write. Quanto aos programas correntes para os IBM PC, o Amstrad PC1512 possui um dos mais altos graus de compatibilidade, desde o «Flight Simulator», da Microsoft, até ao «Lotus 1-2-3» com ou sem «Sidekick», à «Database III» e ao «Open Acess». A única excepção são os programas que usam o sistema EGA de gráficos de muito alta resolução. A Amstrad propõe também uma boa gama de programas especiais para o PC1512, em particular uma versão melhorada do «Wordstar».

A designação de «PC», que de princípio se aplicava aos computadores que permitiam o tratamento pessoal (doméstico, se necessário) de problemas que antes implicavam o recurso aos computadores das empre-

sas, passou a ter um significado mais vasto quando eles próprios começaram a ser integrados em redes locais, permitindo aos utilizadores não só compartilhar vários recursos entre eles como também realizar tarefas independentes. É em utilizações dessa natureza que as qualidades do Amstrad PC1512 mais se evidenciam.

***

APÊNDICE A
MAPA DE MEMÓRIA



APÊNDICE B

CÓDIGOS DE ERRO E DEFINIÇÕES

UNEXPECTED NEXT
Foi encontrada uma instrução NEXT à qual não corresponde qualquer FOR.

SYNTAX ERROR
A linha de programa não tem sentido de acordo com a BASIC do Amstrad. Verifique bem o que escreveu quando ocorrer este erro.

UNEXPECTED RETURN
Semelhante a UNEXPECTED NEXT; o computador encontrou uma instrução RETURN à qual não corresponde nenhuma GOSUB.

DATA EXHAUSTED
O computador executou uma instrução READ, mas não encontrou dados em número suficiente.

IMPROPER ARGUMENT
Parte da instrução está errada — o valor usado como parâmetro não corresponde à gama permitida.

OVERFLOW
Uma operação numérica produziu um valor superior a 1,7 elevado à potência 38.

MEMORY FULL
É um erro muito fácil de compreender. O programa é demasiado grande para caber na memória do computador. Se tal acontece e o programa é apenas um pouco maior do que o possível para o Amstrad, elimine quaisquer espaços supérfluos, instruções REM ou quaisquer instruções que não afectem a lógica do programa.

LINE DOES NOT EXIST
Normalmente, uma GOTO ou GOSUB tentaram enviar a execução para um número de linha que não existe.

SUBSCRIPT OUT OF RANGE
Verifique os seus *arrays*, dado que não existe o elemento indicado na instrução que produz o erro.

ARRAY ALREADY DIMENSIONED
O seu programa tenta redimensionar um *array* que já existe.

DIVISION BY ZERO
Não se pode dividir por zero em nenhuma situação. SC=1000/INT (P) pode produzir este erro se P for inferior a um.

INVALID DIRECT COMMAND
Alguns dos comandos Amstrad só podem ser usados no interior de um programa, e não em modo directo. É o caso de DEF FN, por exemplo.

TYPE MISMATCH
Não se podem misturar valores de cadeia e numéricos — se se tentar, obtém-se este erro.

STRING SPACE FULL
Foram criadas tantas cadeias que já não há espaço para mais.

STRING TOO LONG
As cadeias só podem ter um comprimento de 255 caracteres — se for necessária uma maior, utilize várias cadeias e some-as.

STRING EXPRESSION TOO COMPLEX
As expressões de cadeia podem produzir valores de cadeia intermédios. Se o número destes exceder um limite razoável, ocorre este erro.

CANNOT CONTINUE
Se um programa foi parado por uma instrução STOP ou carregado duas vezes em ESC, e alterado de algum modo, não pode ser continuado com uma ordem CONT.

UNKNOWN USER FUNCTION
Foi referida uma função que ainda não foi definida usando DEF FN.

RESUME MISSING

O programa atingiu o seu final ainda no interior de uma rotina ON ERROR.

UNEXPECTED RESUME

A ordem RESUME só pode ser usada no final de uma rotina ON ERROR; se for usada em qualquer outro ponto de um programa, produz este erro.

DIRECT COMMAND FOUND

Foi encontrada uma linha de programa sem qualquer número de linha quando o computador tentava carregar um programa.

OPERAND MISSING

O computador encontrou uma expressão em BASIC que não se encontra completa. Se escrevermos LOAD e carregarmos em ENTER sem usar aspas a seguir a LOAD, obtemos este erro.

LINE TOO LONG

Uma linha que fique demasiado grande quando convertida para a forma da linguagem BASIC interna ao computador produzirá este erro.

EOF MET

O computador tenta ler para além do final de um ficheiro quando carrega de *cassette*.

FILE TYPE ERROR

Existem vários tipos de ficheiro em *cassette*, e cada um deles requer comandos específicos. Se se usar a ordem errada para o ficheiro considerado obtém-se este erro.

NEXT MISSING

O computador encontrou uma instrução NEXT à qual não corresponde qualquer instrução FOR.

FILE ALREADY OPEN

Foi criado um ficheiro novo antes de o ficheiro anterior ter sido fechado.

UNKNOWN COMMAND

O computador não consegue encontrar um periférico ao qual se aplica a instrução BASIC encontrada.

WEND MISSING

O computador encontrou uma instrução WHILE à qual não corresponde uma WEND.

UNEXPECTED WEND

É essencialmente o erro oposto ao anterior — foi encontrada uma WEND à qual não corresponde uma WHILE.

ÍNDICE

NOTA DA EDIÇÃO PORTUGUESA 7
AGRADECIMENTOS 9
PREFÁCIO 11
1. COMO FUNCIONA O SEU COMPUTADOR? 13
Lógico, meu caro Watson 14
2. PRIMEIROS PASSOS NA PROGRAMAÇÃO 16
Linhas 16
PRINT, LIST, NEW, RUN 17
TAB e LIST 19
Variáveis 21
Cadeias 22
3. MAIS PROGRAMAÇÃO 24
Corrida de cavalos 25
Algumas instruções novas 28
4. CICLOS 33
FOR/NEXT... STEP 35
WHILE e WEND 37
Código 37
5. CONTINUANDO A APRENDER 39
Linhas com várias instruções 39
Algumas funções 40
Funções matemáticas 41
Funções «figurativas» 42
Definição de funções pelo utilizador 45
6. READ/DATA E RESTORE 47
7. NÚMEROS ALEATÓRIOS E DECISÕES 54
Lançamento de dados 56
Decisões, decisões 58
Comparações 58

8. TRATAMENTO DE CADEIAS 60
Comandos específicos 62
Ordenação de cadeias 64
Divisão de cadeias 66
9. ENTRADAS DE VÁRIOS TIPOS 67
Controlo por *joystick* 68
Entradas «artísticas» 69
Outros comandos de teclas 71
10. COMANDOS DE AUXÍLIO À PROGRAMAÇÃO 73
EDIT e montagem de linhas 74
11. O AMSTRAD E A SUA UTILIDADE 76
Ficheiros de dados 78
Gravar um ficheiro 78
Carregar um ficheiro 78
Base de dados 78
Sugestões 80
12. USO DE DISCOS 81
Os discos 81
AMSDOS 82
Cópia de programas de um disco para outro 85
Verificar os discos copiados 85
Logo 87
Primitivas Logo 88
13. JOGOS! JOGOS! JOGOS! 89
Madrugada vermelha 89
Na bolsa 91
Campo de batalha 92
Letras 96
Outro tijolo na parede 98
Cavaleiros 99
14. TRATAMENTO DE TEXTO 103
Máquina de escrever 103
15. UMA EXPLICAÇÃO GRÁFICA 107
Cor, modos e comandos de texto 107
Um arco-íris 109
Gráficos definidos pelo utilizador 114
Deseja outro tipo de letra? 119
Instruções de alta resolução 120
Desenho e teste 123
TAG e TAGOFF, XPOS e YPOS 125
A janela gráfica 127
Animação 128

16. SONS 134
Volume 139
Componha as suas músicas 139
Sincronização 148
17. RUDIMENTOS DE CÓDIGO-MÁQUINA 152
18. INTERRUPÇÕES DE SISTEMA 154
A instrução AFTER 154
TIME 156
19. JANELAS 159
20. AS INSTRUÇÕES *ON ERROR* E *ON BREAK* 163
21. INSTRUÇÕES DIVERSAS 166
O Amstrad e uma impressora 167
22. O AMSTRAD CPC 6128 168
23. COMO ESCREVER PROGRAMAS DE UMA FORMA MAIS PROFISSIONAL 170
Alguns truques de programação 170
AMSTRAD PCW8256/8512 173
AMSTRAD PC1512 175
APÊNDICE A — MAPA DE MEMÓRIA 178
APÊNDICE B — CÓDIGOS DE ERRO E DEFINIÇÕES 179

Este guia de programação do Amstrad é dirigido a todos os utilizadores de computadores Amstrad CPC 464, 664 e 6128, bem como dos novíssimos PCW 8256/8512 e PC 1512. Neste volume pode o leitor aprender a lidar com o seu Amstrad como um verdadeiro profissional. Pode aprender coisas úteis e interessantes como a dominar perfeitamente o manejo da linguagem Basic, melhorar os seus próprios programas, escrever cartas, resolver problemas, gerir as suas finanças, ensinar crianças e até a jogar com o computador, passando horas de agradável entretenimento. Esta edição portuguesa é enriquecida com dois capítulos da autoria de Eurico da Fonseca sobre os PCW 8256/8512 e PC 1512, acompanhando os mais recentes desenvolvimentos desta linha de computadores.

EDITORIAL PRESENÇA

Guia de Programação do Amstrad